Supplementary Material
MATLAB Code for Numerical Examples in Hill paper
"On the Oval Shapes of Beach Stones"

List of programs

1. Aristotle.m
2. New CSF Semi Implicit 6.m
3. DiscretizedStones.m
4. may2waves.m
5. PolygonSlicing3D.m
6. Numerical Solve Curve 2.m

1 Aristotle.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Aristotle.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [matrix1,matrix2] = Aristotle(gamma01,gamma02,L,M,N,Frames,fixOrigin)

% DESCRIPTION: This an implementation of curve shortening flow in the
% plane. This function only does shortening with the formula dh/dt = -h^2.

% INPUT: (gamma01,gamma02) is an inline parametric plane curve with
% period L. The approximate solution we will generate is defined on
% [0,Tmax] x [0,L]. We will use M subintervals for [0,Tmax] and N
% subintervals for [0,L].

% OUTPUT: A sequence of plots of the approximate solution curves

% "delta p"
k=L/N;

% Intialize gamma matrix: gamma1(i,j) approximates the true solution
% gamma1(ti,pj), where ti=i*h, i=0,...,M and pj=j*k, j=0,...,N, and similar
% for gamma2(i,j)
gamma1=zeros(M*15+1,N+1);
gamma2=zeros(M*15+1,N+1);

% Initial Condition
p=(0:k:L);
gamma1(1,:)=gamma01(p);
gamma2(1,:)=gamma02(p);

Tmax=.99*polyarea(gamma01(p),gamma02(p))/(2*pi);

% "delta t"
```

```
t=Tmax/(M);

polyin = polyshape(gamma1(1, 1:N),gamma2(1, 1:N));

[centerOfGravX, centerOfGravY] = centroid(polyin);

gamma1(1, :) = gamma1(1, :) - centerOfGravX;
gamma2(1, :) = gamma2(1, :) - centerOfGravY;

% Plot initial condition gamma0
plot(gamma1(1, :),gamma2(1, :));
pause(0.001)

% Setting the axis of the figure
axis([min(gamma1(1, :)) max(gamma1(1, :)) min(gamma2(1, :)) max(gamma2(1, :))])
axis equal

warningIdentifier = -1;

% switch to true for an animated graph
animated = false;

if animated
    writerObj = VideoWriter('example.avi');
    writerObj.FrameRate = 60;
    open(writerObj);

    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');
else
    hold on
end

maxNorm = max(vecnorm([ gamma1(1,:); gamma2(1,:) ]));

% Explicit Scheme
for i=1:(M*15)
    polyin = polyshape(gamma1(i, 1:N),gamma2(i, 1:N));

    if warningIdentifier == -1
        [msgStr, msgId] = lastwarn;
        warningIdentifier = msgId;
        warning('off', msgId)
        warning(msgStr)
    end

    [centerOfGravX, centerOfGravY] = centroid(polyin);

    avgRad = sqrt(area(polyin)/pi);

    % shape coordinates with center of gravity at the origin
    centeredX = gamma1(i, :) - centerOfGravX;
    centeredY = gamma2(i, :) - centerOfGravY;

    norms = vecnorm([ centeredX; centeredY ]);

    % if you want to evolve according to dh/dt = -h, uncomment the two
```

```
    % lines below and comment the two lines after.
%     gamma1(i+1, :) = centeredX - centeredX.*min([t/(avgRad^2); 1]);
%     gamma2(i+1, :) = centeredY - centeredY.*min([t/(avgRad^2); 1]);
    gamma1(i+1, :) = centeredX - centeredX.*min([t*norms/(avgRad^3); ones(1, N+1)]);
    gamma2(i+1, :) = centeredY - centeredY.*min([t*norms/(avgRad^3); ones(1, N+1)]);

    % if we don't want it so that we fix the origin every time, move the
    % shape that we found through evolution of the centered shape back to
    % the original center of gravity so it's like we used distances to the
    % original center of gravity instead
    if ~fixOrigin
        gamma1(i+1, :) = gamma1(i+1, :) + centerOfGravX;
        gamma2(i+1, :) = gamma2(i+1, :) + centerOfGravY;
    end


    if animated
        if (mod(i,Frames)==0)
            %disp(max(gamma2(i+1, :))/max(gamma1(i+1, :)))
            norms = vecnorm([ gamma1(i,:); gamma2(i,:) ]);
            if max(norms) < maxNorm / 200
                disp('broke')
                break
            end
            pause(0.001)
            set(myfig,'XData', gamma1(i+1,:))
            set(myfig,'YData', gamma2(i+1,:))
            frame = getframe(gcf, [50 25 460 380]);
            writeVideo(writerObj,frame);
        end
    else
        if (mod(i,Frames)==0)
            if max(norms) < maxNorm / 200
                disp('broke')
                break
            end
        end

        %if (mod(i,4000)==0)
        %if (mod(i,20000)==0)
        %if i == 40000 || i == 80000 || i == 120000 || i == 160000 || i == 187500 || i == 210000
        %if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 16000
        if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 14329
        %if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 15000
        %if i == 1400 || i == 2700 || i == 4500 || i == 8000 || i == 9800

        %if i == 3300 || i == 6600 || i == 10000 || i == 13300 || i == 21000 || i == 42000
            disp(i)
            plot(gamma1(i+1,:), gamma2(i+1,:))
            pause(0.001)
        end
    end
end

if ~animated
    hold off
end
```

```
if warningIdentifier ~= -1
    warning('on', warningIdentifier)
end

if animated
    close(writerObj);
end

matrix1 = gamma1;
matrix2 = gamma2;

end

%%%%%% Examples of smooth embedded plan curves to evolve via CSF ?? %%%%%%

% Unit circle: x^2+y^2=1
% [m1, m2] = Aristotle(@cos,@sin,2*pi,20000,100,50,false);

% Ellipse: (x/3)^2+(y/2)^2=1
% [m1, m2] = Aristotle(@(p)2*cos(p),@(p)3*sin(p),2*pi,15000,200,50,false);

% Ellipse, different form: x^2+(y/b)^2=1, b = 0.9
% [m1, m2] = Aristotle(@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*cos(p),
@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*sin(p),2*pi,15000,200,50,false);

% Egg-shape: k(cos^2(theta) + a*cos(theta) + b), b = 0.9
% [m1, m2] = Aristotle(@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*cos(p), @(p)(cos(p).^2 +
1.04*cos(p) + 2.3).*sin(p),2*pi,15000,200,50,false);

% Triangle ISOC
% [m1, m2] = Aristotle(@(p)radiusisocx(p), @(p)radiusisocy(p),2*pi,15000,200,50,false);

% Circle dent
% [m1, m2] = Aristotle(@(p)circledentx(p), @(p)sin(p),2*pi,15000,200,50,false);

% Non-embedded curve:
% [m1, m2] = Aristotle(@(p)(cos(2*p).*cos(p)),@(p)(cos(2*p).*sin(p)),2*pi,15000,200,50,false);

% Folium:
% [m1, m2] = Aristotle(@(p)(-(cos(p).^3).*cos(p)),@(p)(-(cos(p).^3).*sin(p)),2*pi,15000,200,50,false);
% Row 7598

% Limcon of Pascal: r = 3/2 + cos(theta)
% Aristotle(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,15000,200,50,false);

% f = @() Aristotle(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,15000,200,50,false);
% timeit(f)

% Lame curve: (x/3)^1/4+(y/4)^1/4=1
% [m1, m2] = Aristotle(@(p)(abs(cos(p)).^(1/2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(1/2) * 4 .* sign(sin(p))),2*pi,15000,200,50,false);

% Another Lame curve:
% [m1, m2] = Aristotle(@(p)(abs(cos(p)).^(2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(2) * 4 .* sign(sin(p))),2*pi,15000,200,50,false);
```

```
% Polygon:
% [m1, m2] = Aristotle(@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* cos(p)),
@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* sin(p)),2*pi,15000,200,50,false);

% Non-embedded Lame curve:
% [m1, m2] = Aristotle(@(p)(abs(cos(p)).^(4) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(4) * 4 .* sign(sin(p))),2*pi,15000,200,50,false);

% Triangle SCAL
% [m1, m2] = Aristotle(@(p)trianglex(p), @(p)triangley(p),2*pi,15000,200,50,false);
```

2 New_CSF_Semi_Implicit_6.m

```
%%%%%%%%%%%%%%%%%%%%%%%%% New_CSF_Semi_Implicit_6.m %%%%%%%%%%%%%%%%%%%%%%%%%

function [matrix1,matrix2] = New_CSF_Semi_Implicit_6(gamma01,gamma02,L,M,N,Frames,animated,normal,
expNum,useSupport,fixOrigin)

% DESCRIPTION: This an implementation of curve shortening flow in the
% plane. We employ a semi-implicit finite difference scheme. This
% method will take a C^1, closed, embedded plane curve and deform it
% according to curve shortening flow for the life of the flow (see Tmax
% below)

% INPUT: (gamma01,gamma02) is an inline parametric plane curve with
% period L. The approximate solution we will generate is defined on
% [0,Tmax] x [0,L]. We will use M subintervals for [0,Tmax] and N
% subintervals for [0,L]. animated - determine if we want the evolution to
% be animated or not, normal - if yes, evolve based on curvature, if not,
% evolve based on h^expNum in the denominator.

% OUTPUT: A sequence of plots of the approximate solution curves


% "delta p"
k=L/N;

% Intialize gamma matrix: gamma1(i,j) approximates the true solution
% gamma1(ti,pj), where ti=i*h, i=0,...,M and pj=j*k, j=0,...,N, and similar
% for gamma2(i,j)
gamma1=zeros(M*15+1,N+1);
gamma2=zeros(M*15+1,N+1);

% Initial Condition
p=(0:k:L);
gamma1(1,:)=gamma01(p);
gamma2(1,:)=gamma02(p);

% Using polygonal Formula for Tmax
Tmax=.99*polyarea(gamma01(p),gamma02(p))/(2*pi);

% "delta t"
t=Tmax/M;

polyin = polyshape(gamma1(1, 1:N),gamma2(1, 1:N));

[centerOfGravX, centerOfGravY] = centroid(polyin);

gamma1(1, :) = gamma1(1, :) - centerOfGravX;
gamma2(1, :) = gamma2(1, :) - centerOfGravY;

% Plot initial condition gamma0
myfig=plot(gamma1(1, :),gamma2(1, :));

% Setting the axis of the figure
axis([min(gamma1(1, :)) max(gamma1(1, :)) min(gamma2(1, :)) max(gamma2(1, :))])
axis equal
pause(0.001)
```

```
warningIdentifier = -1; % boolean for making polygon warnings not print

% setup plots based on whether the graph is animated
if animated
    writerObj = VideoWriter('example.avi');
    writerObj.FrameRate = 60;
    open(writerObj);

    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');
else
    hold on
end

maxNorm = max(vecnorm([ gamma1(1,:); gamma2(1,:) ]));

% Semi-Implicit Scheme
for i=1:(M*15)

    polyin = polyshape(gamma1(i, 1:N),gamma2(i, 1:N));

    % conditional statement so warnings are not always printed
    if warningIdentifier == -1
        [msgStr, msgId] = lastwarn;
        warningIdentifier = msgId;
        warning('off', msgId)
        warning(msgStr)
    end

    avgRad = sqrt(area(polyin)/pi);

    [centerOfGravX, centerOfGravY] = centroid(polyin);

    % shape coordinates with center of gravity at the origin
    centeredX = gamma1(i, :) - centerOfGravX;
    centeredY = gamma2(i, :) - centerOfGravY;

    % if we want a fixed origin, set shape to the values above
    if fixOrigin
        gamma1(i, :) = centeredX;
        gamma2(i, :) = centeredY;
    end

    % to calculate support vectors, we use values from the centered shape
    if useSupport
        norms = vecnorm([ centeredX; centeredY ]);
        unitvecs = [(centeredX./norms).', (centeredY./norms).'];
        dotproducts = unitvecs * [centeredX; centeredY];
        rad = max(dotproducts.');
    end

    triDiagMat = zeros(N+1);

    h = zeros(N+1, 1);

    for j=2:N+1
```

```
    h(j) = norm([ (gamma1(i,j)-gamma1(i,j-1)) (gamma2(i,j)-gamma2(i,j-1)) 10^(-8)]);

end

h(1) = h(N+1);

for j=1:N

    if normal
        distanceDenom = 1;
    else
        if useSupport
            distanceDenom = ((rad(j)/avgRad)^expNum);
        else
            distanceDenom = ((norm([centeredX(j) centeredY(j)])/avgRad)^expNum);
        end
    end

    prev = j-1;
    if prev == 0
        prev = N;
    end

    triDiagMat(j, prev) = -1/h(j) / distanceDenom;
    triDiagMat(j, j+1) = -1/h(j+1) / distanceDenom;
    triDiagMat(j, j) = 1/(2*t)*(h(j) + h(j+1)) + (1/h(j) + 1/h(j+1)) / distanceDenom;

end

triDiagMat(N+1, N) = triDiagMat(1, N);
triDiagMat(N+1, 2) = triDiagMat(1, 2);
triDiagMat(N+1, N+1) = triDiagMat(1, 1);

vectorComplex = zeros(N+1, 1);

for j=1:N

    vectorComplex(j) = 1/(2*t)*complex(gamma1(i, j), gamma2(i, j)) * (h(j) + h(j+1));

end

vectorComplex(N+1) = vectorComplex(1);

newVecComplex = triDiagMat\vectorComplex;

gamma1(i+1, :) = real(newVecComplex);
gamma2(i+1, :) = imag(newVecComplex);

if animated
    if (mod(i,Frames)==0)
        %disp(max(gamma2(i+1, :))/max(gamma1(i+1, :)))
        norms = vecnorm([ centeredX; centeredY ]);
        if max(norms) < maxNorm / 200
            disp('broke')
            break
        end
```

```
            pause(0.001)
            set(myfig,'XData', gamma1(i+1,:))
            set(myfig,'YData', gamma2(i+1,:))
            frame = getframe(gcf, [50 25 460 380]);
            writeVideo(writerObj,frame);
        end
    else
        if (mod(i,Frames)==0)
            norms = vecnorm([ centeredX; centeredY ]);
            if max(norms) < maxNorm / 200
                disp('broke')
                break
            end
        end
        %if (mod(i,1000)==0)

        %if i == 4650 || i == 8300 || i == 13200 || i == 20000 || i == 26000 || i == 33000
        %if i == 40000 || i == 80000 || i == 120000 || i == 160000 || i == 187500 || i == 205000
        %if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 16000
        %if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 15700
        %if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 15300
        if i == 2000 || i == 4000 || i == 7000 || i == 12000 || i == 15000
        %if i == 1500 || i == 4000 || i == 7000 || i == 11000 || i == 13500 || i == 15400 || i == 16000
            disp(i)
            plot(gamma1(i+1,:), gamma2(i+1,:))
            pause(0.001)
        end
    end
end

if ~animated
    hold off
else
    close(writerObj);
end

if warningIdentifier ~= -1
    warning('on', warningIdentifier)
end

matrix1 = gamma1;
matrix2 = gamma2;

end

%%%%%% Examples of smooth embedded plan curves to evolve via CSF ?? %%%%%%

% Unit circle: x^2+y^2=1
% [m1, m2] = New_CSF_Semi_Implicit_6(@cos,@sin,2*pi,20000,100,50,false,false,3,false,false);

% Ellipse: (x/3)^2+(y/2)^2=1
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)2*cos(p),@(p)3*sin(p),2*pi,15000,200,50,
false,false,3,false,false);

% Ellipse, different form: x^2+(y/b)^2=1, b = 0.9
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*cos(p),
@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*sin(p),2*pi,15000,200,50,false,false,3,false,false);
```

```
% Egg-shape: k(cos^2(theta) + a*cos(theta) + b), b = 0.9
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*cos(p),
@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*sin(p),2*pi,15000,200,50,false,false,3,false,false);

% Triangle ISOC
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)radiusisocx(p), @(p)radiusisocy(p),2*pi,15000,200,50,
false,false,3,false,false);

% Circle dent
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)circledentx(p), @(p)sin(p),2*pi,15000,
200,50,false,false,3,false,false);

% Non-embedded curve:
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(cos(2*p).*cos(p)),@(p)(cos(2*p).*sin(p)),2*pi,
15000,200,50,false,false,3,false,false);

% Folium:
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(-(cos(p).^3).*cos(p)),@(p)(-(cos(p).^3).*sin(p)),2*pi,
15000,200,50,false,false,3,false,false);
% Row 7598

% Limcon of Pascal: r = 3/2 + cos(theta)
% New_CSF_Semi_Implicit_6(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,
15000,200,50,false,false,3,false,false);

% f = @() New_CSF_Semi_Implicit_6(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,
15000,200,50,false,false,3,false,false);
% timeit(f)

% Lame curve: (x/3)^1/4+(y/4)^1/4=1
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(abs(cos(p)).^(1/2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(1/2) * 4 .* sign(sin(p))),2*pi,15000,200,50,false,false,3,false,false);

% Another Lame curve:
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(abs(cos(p)).^(2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(2) * 4 .* sign(sin(p))),2*pi,15000,200,50,false,false,3,false,false);

% Polygon:
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* cos(p)),
@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* sin(p)),2*pi,15000,200,50,false,
false,3,false,false);

% Non-embedded Lame curve:
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)(abs(cos(p)).^(4) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(4) * 4 .* sign(sin(p))),2*pi,15000,200,50,false,false,3,false,false);


% Triangle SCAL
% [m1, m2] = New_CSF_Semi_Implicit_6(@(p)trianglex(p),
@(p)triangley(p),2*pi,15000,200,50,false,false,3,false,false);
```

3 DiscretizedStones.m

```
function DiscretizedStones(gamma01,gamma02,L,M,N,Frames,angleChoice,probabilistic,nChoice,animated)

delta = 0.01;

% "delta p"
k=L/N;

% Initial Condition
p=(0:k:L);
gamma1=gamma01(p);
gamma2=gamma02(p);

xValues = gamma1(1, 1:N);
yValues = gamma2(1, 1:N);
polyin = polyshape(xValues,yValues);

[centerOfGravX, centerOfGravY] = centroid(polyin);
gamma1 = gamma1 - centerOfGravX;
gamma2 = gamma2 - centerOfGravY;

% Plot initial condition gamma0
myfig=plot(gamma1,gamma2);
pause(0.001)

% Setting the axis of the figure
axis([min(gamma1) max(gamma1) min(gamma2) max(gamma2)])
axis equal

set(myfig,'XData', gamma1)
set(myfig,'YData', gamma2)
pause(0.001)

% setup plots based on whether the graph is animated
if animated
    writerObj = VideoWriter('example.avi');
    writerObj.FrameRate = 60;
    open(writerObj);

    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');
else
    hold on
end

maxNorm = max(vecnorm([ gamma1; gamma2]));

theta = 0;

% setup values for later calculating support function values if needed
int = 0:2*pi/200:2*pi-2*pi/200;
unitcirc = zeros(200, 2);
unitcirc(:, 1) = cos(int);
unitcirc(:, 2) = sin(int);
if strcmp(angleChoice, 'cubesupii')
    num = 2^nChoice;
```

```
    int = 0:2*pi/num:2*pi-2*pi/num;
    unitcirc = zeros(num, 2);
    unitcirc(:, 1) = cos(int);
    unitcirc(:, 2) = sin(int);
end

numFlips = 1;

tic

for i=1:M

    origArea = area(polyin);
    [centerOfGravX, centerOfGravY] = centroid(polyin);
    xValues = xValues - centerOfGravX;
    yValues = yValues - centerOfGravY;
    polyin = polyshape(xValues,yValues);

    maxX = max(xValues);
    minX = min(xValues);

    maxY = max(yValues);
    minY = min(yValues);

    disp = 0;

    if strcmp(angleChoice, 'cubesupiii') || strcmp(angleChoice, '2.5supiii')
        a = bitshift(randi([0, 2^32-1]), -5);
        b = bitshift(randi([0, 2^32-1]), -6);
        random = (a * 67108864.0 + b) / 9007199254740992.0;
        disp = random * 2 * pi;
        num = 2^nChoice;
        int = 0:2*pi/num:2*pi-2*pi/num;
        int = int + disp;
        unitcirc = zeros(num, 2);
        unitcirc(:, 1) = cos(int);
        unitcirc(:, 2) = sin(int);
    end

    if strcmp(angleChoice, 'random')
        a = bitshift(randi([0, 2^32-1]), -5);
        b = bitshift(randi([0, 2^32-1]), -6);
        random = (a * 67108864.0 + b) / 9007199254740992.0;
        theta = random * 2 * pi;
    elseif strcmp(angleChoice, 'randomii')
        theta = (randsample(2^nChoice, 1) - 1) * 2*pi/2^nChoice;
    elseif strcmp(angleChoice, 'cubesupii')
        dotproducts = unitcirc * [xValues; yValues];
        supdist = max(dotproducts.');
        supdist = supdist.^3;
        theta = (randsample(2^nChoice, 1, true, 1./supdist)-1) * 2*pi/2^nChoice;
    elseif strcmp(angleChoice, 'cubesupiii') % uniform displacements, random drawings
        dotproducts = unitcirc * [xValues; yValues];
        supdist = max(dotproducts.');
        supdist = supdist.^3;
        theta = (randsample(2^nChoice, 1, true, 1./supdist)-1) * 2*pi/2^nChoice+disp;
    elseif strcmp(angleChoice, '2.5supiii') % uniform displacements, random drawings
```

```
    dotproducts = unitcirc * [xValues; yValues];
    supdist = max(dotproducts.');
    supdist = supdist.^2.5;
    theta = (randsample(2^nChoice, 1, true, 1./supdist)-1) * 2*pi/2^nChoice+disp;
elseif strcmp(angleChoice, 'deterministic')
    theta = theta + 2 * pi / 100;
elseif strcmp(angleChoice, 'inversesupport')
    dotproducts = unitcirc * [xValues; yValues];
    supdist = max(dotproducts.');
    theta = randsample(200, 1, true, 1./supdist) / 100 * pi;
elseif strcmp(angleChoice, 'cubesupport')
    dotproducts = unitcirc * [xValues; yValues];
    supdist = max(dotproducts.');
    supdist = supdist.^3;
    theta = randsample(200, 1, true, 1./supdist) / 100 * pi;
elseif strcmp(angleChoice, 'nonneedle')
    p1 = maxX^2 - 3*maxX + 3;
    p2 = maxY^2 - 3*maxY + 3;
    p3 = abs(minX)^2 - 3*abs(minX) + 3;
    p4 = abs(minY)^2 - 3*abs(minY) + 3;
    theta = (randsample(4, 1, true, [p1, p2, p3, p4]) - 1) * pi/2;
elseif strcmp(angleChoice, 'a4')
    theta = (randsample(4, 1) - 1) * pi/2;
elseif strcmp(angleChoice, 'a5')
    theta = (randsample(4, 1, true, 1./[maxX maxY abs(minX) abs(minY)]) - 1) * pi/2;
else
    error('angleChoice must be a valid stochastic category')
end

% calcs for reformating shape for valid area calculation in next step
direction = [cos(theta) sin(theta)];
direction_scaled = direction * norm([(maxX - minX) (maxY - minY)]);

shape = [xValues; yValues];
intersections = intersect(polyin, [[0 0]; [direction_scaled(2) -direction_scaled(1)]]);
cutPoint1 = intersections(1, :);
if isequal(cutPoint1, [0 0])
    cutPoint1 = intersections(2, :);
end
cutPoint1 = cutPoint1.';

intersections = intersect(polyin, [[0 0]; [-direction_scaled(2) direction_scaled(1)]]);
cutPoint2 = intersections(1, :);
if isequal(cutPoint2, [0 0])
    cutPoint2 = intersections(2, :);
end
cutPoint2 = cutPoint2.';

[~, index1] = min(vecnorm(shape - cutPoint1));
[~, index2] = min(vecnorm(shape - cutPoint2));

if index1 > index2
    xValues = [xValues(index1:numel(xValues)) xValues(1:index1-1)];
    yValues = [yValues(index1:numel(yValues)) yValues(1:index1-1)];
end

% x value of points, evaluated on the perpendicular of cut direction,
```

```
% find the max distance in that direction
dots = direction * [xValues; yValues];
xDirections = [direction(2) -direction(1)] * [xValues; yValues];
maxD = max(dots);

% determine amount of area to cut away, if probabilistic, sample from
% an exponential distribution
deltaTimesArea = delta;
if probabilistic
     deltaTimesArea = exprnd(2)*2.5*delta;
end

% iterate through all distances possible, find cuts that gives
% deltaTimesArea proportion of area cut away
for d=maxD:-maxD/10000:0

     % basically, y values are distances in theta direction away from
     % the perpendicular cut to points, combined with x values, can be
     % used to calculate area of the cut efficiently

     ys = dots - d;
     positiveYs = ys > 0;
     indices = find(positiveYs);
     if numel(indices) < 1
         continue
     end

     firstIndex = indices(1);
     lastIndex = indices(numel(indices));

     prevIndex = firstIndex - 1;

     if firstIndex == 1
         prevIndex = numel(ys);
     end

     y1 = ys(prevIndex);
     y2 = ys(firstIndex);
     x1 = xDirections(prevIndex);
     x2 = xDirections(firstIndex);
     newX1 = -y1/(y2-y1)*(x2-x1)+x1;

     forwardIndex = lastIndex + 1;

     if lastIndex == numel(ys)
         forwardIndex = 1;
     end

     y1 = ys(lastIndex);
     y2 = ys(forwardIndex);
     x1 = xDirections(lastIndex);
     x2 = xDirections(forwardIndex);
     newX2 = -y2/(y1-y2)*(x1-x2)+x2;

     xs = xDirections(positiveYs);
     xs = [newX1 xs newX2];
     ys = ys(positiveYs);
```

```
        ys = [0 ys 0];

        printArea = abs(trapz(xs, ys));

        if deltaTimesArea*origArea <= printArea
            break
        end
    end

%     i
%     d
%     origArea
%     printArea
%
%     if theta == 2*pi/100
%         pause(10)
%     end

    % make the cut, produce new polygon
    cutHere = direction * d;
    intersectpolygon = [cutHere + [-direction_scaled(2) direction_scaled(1)]; ...
        cutHere + [-direction_scaled(2) direction_scaled(1)] - 2*direction_scaled; ...
        cutHere + [direction_scaled(2) -direction_scaled(1)] - 2*direction_scaled; ...
         cutHere + [direction_scaled(2) -direction_scaled(1)]];
    intersectpolygon = polyshape(intersectpolygon);
    polyout = intersect(polyin, intersectpolygon);

    polyin = polyout;

    % make counterclockwise
    xValues = flip(polyin.Vertices(:, 1).');
    yValues = flip(polyin.Vertices(:, 2).');

    % flip again if not counterclockwise
    if mod(atan2d(yValues(2), xValues(2)) - atan2d(yValues(1), xValues(1)), 360) >= 180
        xValues = flip(xValues);
        yValues = flip(yValues);
        polyin = polyshape(xValues, yValues);
        disp("found " + numFlips)
        disp(i)
        numFlips = numFlips + 1;
    end

    if animated
        if (mod(i,Frames)==0)
            pause(0.001)
            set(myfig,'XData', [xValues, xValues(1)])
            set(myfig,'YData', [yValues, yValues(1)])
            %frame = getframe;
            %frame = getframe(gcf, [74 50 435 350]);
            % axesssssssss
            frame = getframe(gcf, [50 25 460 380]);
            writeVideo(writerObj,frame);
        end
    else
        if (mod(i,Frames)==0)
            norms = vecnorm([ xValues; yValues ]);
```

```
            if i == 505
            %if max(norms) < maxNorm / 200
                disp('broke')
                break
            end
        end
        if (mod(i,100)==0)
        %if i == 90 || i == 200 || i == 300 || i == 450 || i == 700
            plot([xValues, xValues(1)], [yValues, yValues(1)])
            pause(0.001)
        end
    end

    if i == 1000
        toc
    end
end

% if animated or not, make final adjustments, close writer objects
if ~animated
    hold off
else
    close(writerObj);

    plot([xValues, xValues(1)],[yValues, yValues(1)]);
    pause(0.001)

    % Setting the axis of the figure
    axis([min(xValues) max(xValues) min(yValues) max(yValues)])
    axis equal
    pause(0.001)
end

beep

end

%%%%%% Examples of curves to evolve via Discretized Shaving of Stones %%%%%%

% Unit circle: x^2+y^2=1
% DiscretizedStones(@cos,@sin,2*pi,20000,100,5,'randomii',false,10,true);

% Ellipse: (x/3)^2+(y/2)^2=1
% DiscretizedStones(@(p)2*cos(p),@(p)3*sin(p),2*pi,15000,200,5,'randomii',false,10,true);

% Ellipse, different form: x^2+(y/b)^2=1, b = 0.9
% DiscretizedStones(@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*cos(p),
@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*sin(p),2*pi,15000,200,5,'randomii',false,10,true);

% Egg-shape: k(cos^2(theta) + a*cos(theta) + b), b = 0.9
% DiscretizedStones(@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*cos(p),
@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*sin(p),2*pi,15000,200,5,'randomii',false,10,true);

% Triangle ISOC
% DiscretizedStones(@(p)radiusisocx(p), @(p)radiusisocy(p),2*pi,15000,200,5,'randomii',false,10,true);

% Circle dent
```

```
% DiscretizedStones(@(p)circledentx(p), @(p)sin(p),2*pi,15000,200,5,'randomii',false,10,true);

% Non-embedded curve:
% DiscretizedStones(@(p)(cos(2*p).*cos(p)),@(p)(cos(2*p).*sin(p)),2*pi,15000,200,5,'randomii',
false,10,true);

% Folium:
% DiscretizedStones(@(p)(-(cos(p).^3).*cos(p)),@(p)(-(cos(p).^3).*sin(p)),2*pi,15000,200,5,
'randomii',false,10,true);
% Row 7598

% Limcon of Pascal: r = 3/2 + cos(theta)
% DiscretizedStones(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,15000,200,5,
'randomii',false,10,true);

% f = @() New_CSF_Semi_Implicit_6(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),
2*pi,15000,200,5,'randomii',false,10,true);
% timeit(f)

% Lame curve: (x/3)^1/4+(y/4)^1/4=1
% DiscretizedStones(@(p)(abs(cos(p)).^(1/2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(1/2) * 4 .* sign(sin(p))),2*pi,15000,200,5,'randomii',false,10,true);

% Another Lame curve:
% DiscretizedStones(@(p)(abs(cos(p)).^(2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(2) * 4 .* sign(sin(p))),2*pi,15000,200,5,'randomii',false,10,true);

% Polygon:
% DiscretizedStones(@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* cos(p)),
@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* sin(p)),2*pi,15000,200,5,
'randomii',false,10,true);

% Non-embedded Lame curve:
% DiscretizedStones(@(p)(abs(cos(p)).^(4) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(4) * 4 .* sign(sin(p))),2*pi,15000,200,5,'randomii',false,10,true);

% Triangle
% DiscretizedStones(@(p)trianglex(p), @(p)triangley(p),2*pi,15000,200,5,'randomii',false,10,true);


% Rectangle
% DiscretizedStones(@(p)rectanglex(p), @(p)rectangley(p),2*pi,15000,200,5,'randomii',false,10,true);
```

4 may2waves.m

```
% Jun 21st email, with Pareto magnitude waves with mean 2

T = 10;
N = 200;

period = 2*pi;

t = 0:T*period/N:T*period;
ut = rand(1, T+1);
xt = (1-ut).^(-0.5);
y = 0.5*xt(floor(t/period)+1).*(sin(t));

plot(t, y)
xlim([min(t) max(t)])
axis equal
pause(0.001)

% May 18th notes, with the sin 2*pi*t and the floors

% T = 7;
% N = 200;
%
% period = 2*pi;
%
% t = 0:T/N:T;
% ut = rand(1, T+1);
% xt = 1./(1-ut).^(1/3);
% y = xt(floor(t)+1).*(sin(t*period));
%
% plot(t, y)
% axis equal
% pause(0.001)

% Some things I tried from the April 22nd iteration, as well as the moving
% average means from my May 3rd email

% T = 3;
% N = 200;
%
% a = 1;
% b = 0.5;
%
% normals = normrnd(0,sqrt(T/N),[1,N]);
% brownian_outputs = cumsum(normals);
% brownian_outputs = [0, brownian_outputs];
% x = 0:T/N:T;
% % y = a*sin(x) + b*sin(brownian_outputs);
% % y = a*(b*sin(brownian_outputs)+1).*sin(x);
% y = a*sin(2*pi*x) + b*cos(brownian_outputs);
% y = movmean(y, 15);
%
% plot(x, y);


% Brownian Magintudes to get waves with closer magnitudes, from my May 22nd
```

```
% email.

% T = 7;
% N = 200;
%
% period = 2*pi;
%
% t = 0:T/N:T;
% normals = normrnd(0,0.7,[1,T+1]);
% brownian_outputs = cumsum(normals);
% brownian_outputs = [0, brownian_outputs];
% xt = (1-abs(cos(brownian_outputs))) + 1.5;
% y = xt(floor(t)+1).*(sin(t*period));
%
% plot(t, y)
```

5 PolygonSlicing3D.m

```
function PolygonSlicing3D(rOrV,thetaOrF,phi,M,Frames,angleChoice,probabilistic,animated,polar)
delta = 0.001;
if probabilistic
    delta = delta * 2;
end
if polar
    r = rOrV;
    theta = thetaOrF;
    width = size(r,2);
    height = size(r,1);

    % all radii for first row and last row must be the same in r
    if ~all(r(1,:) == r(1, 1))
        error('all radii for the first row must be the same in r')
    end
    if ~all(r(height,:) == r(height, 1))
       error('all radii for the last row must be the same in r')
    end

    % Initial Condition
    % P = zeros(width*(height-2)+2,3);
    theta = reshape(theta,[1,width]);
    phi = reshape(phi,[height,1]);

    % P(1, :) = [0 0 r(1, 1)];
    % x = (r(2:height-1, :).*sin(phi(2:height-1))).*cos(theta);
    % y = (r(2:height-1, :).*sin(phi(2:height-1))).*sin(theta);
    % z = r(2:height-1, :).*cos(phi(2:height-1));
    % P(2:width*(height-2)+1, 1) = reshape(x.',[width*(height-2), 1]);
    % P(2:width*(height-2)+1, 2) = reshape(y.',[width*(height-2), 1]);
    % P(2:width*(height-2)+1, 3) = reshape(z.',[width*(height-2), 1]);
    % P(width*(height-2)+2, :) = [0 0 -r(height, 1)];

    X = (r.*sin(phi)).*cos(theta);
    Y = (r.*sin(phi)).*sin(theta);
    Z = r.*cos(phi);

    X = flipud(X);
    Y = flipud(Y);
    Z = flipud(Z);

    %[V, F] = surfToMesh(P(:,1),P(:,2),P(:,3));
    [V, F] = surfToMesh(X,Y,Z);

    C = polyhedronCentroid(V, F);
    X = X - C(1);
    Y = Y - C(2);
    Z = Z - C(3);

    [V, F] = surfToMesh(X,Y,Z);
else
    V = rOrV;
    F = thetaOrF;
    C = polyhedronCentroid(V, F);
```

```
    trans = createTranslation3d(-C(1), -C(2), -C(3));

    [V, F] = transformMesh(V, F, trans);
end

drawMesh(V, F); view(3);

% Setting the axis of the figure
axis([min(V(:, 1)) max(V(:, 1)) min(V(:, 2)) max(V(:, 2)) min(V(:, 3)) max(V(:, 3))])
axis equal

pause(0.001)

origVolume = meshVolume(V, F);

% setup plots based on whether the graph is animated
if animated
    writerObj = VideoWriter('example.avi');
    writerObj.FrameRate = 40;
    open(writerObj);

    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');
else
    hold on
end

maxNorm = max(vecnorm(V.'));

% CREATE 182 lines samples based on uniform spherical coordinates, for
% calculating suport values in that direction

% miniPhi = 10;
% miniWidth = 20;
%
% lessThetas = 0:2*pi/miniWidth:2*pi-2*pi/miniWidth;
% lessPhis = 0:pi/miniPhi:pi;
% lessPhis = reshape(lessPhis,[miniPhi+1,1]);
%
% numLines = miniWidth*(miniPhi-1)+2;
%
% TestLines = zeros(numLines, 3);
%
% TestLines(1, :) = [0 0 1];
% smallX = (ones(miniPhi-1, miniWidth).*sin(lessPhis(2:miniPhi))).*cos(lessThetas);
% smallY = (ones(miniPhi-1, miniWidth).*sin(lessPhis(2:miniPhi))).*sin(lessThetas);
% smallZ = ones(miniPhi-1, miniWidth).*cos(lessPhis(2:miniPhi));
% TestLines(2:numLines-1, 1) = reshape(smallX.',[numLines-2, 1]);
% TestLines(2:numLines-1, 2) = reshape(smallY.',[numLines-2, 1]);
% TestLines(2:numLines-1, 3) = reshape(smallZ.',[numLines-2, 1]);
% TestLines(numLines, :) = [0 0 -1];


% Create 12 EQUALLY SPACED lines based on vertex positions of the
% isocahedron

numLines = 12;
```

```
TestLines = zeros(numLines, 3);
TestLines(1, :) = [0 0 1];
TestLines(2:11, 1) = cos(0:2*pi/10:2*pi-2*pi/10)*cos(atan(0.5));
TestLines(2:11, 2) = sin(0:2*pi/10:2*pi-2*pi/10)*cos(atan(0.5));
TestLines(2:2:10, 3) = sin(atan(0.5));
TestLines(3:2:11, 3) = -sin(atan(0.5));
TestLines(numLines, :) = [0 0 -1];

probabilisticCutRatios = []; % keep track of ratio of volumes cut away

warningIdentifier = -1; % boolean for making polygon warnings not print

tic

for i=1:M
    disp(origVolume)

    % translate shape back to center
    C = polyhedronCentroid(V, F);

    % conditional statement so warnings are not always printed
    if warningIdentifier == -1
        [msgStr, msgId] = lastwarn;
        warningIdentifier = msgId;
        warning('off', msgId)
        warning(msgStr)
    end

    trans = createTranslation3d(-C(1), -C(2), -C(3));
    assignin('base', 'V1', V);
    assignin('base', 'C', C);
    assignin('base', 'F', F)

    [V1, F] = transformMesh(V, F, trans);

    % perform a random 3d rotation on the 12 uniform cutting directions to
    % add a bit of randomness to which of the 12 directions (or more) we
    % pick to cut perpendicular to
    a = 2*pi*rand();
    B = 2*pi*rand();
    y = 2*pi*rand();

    yaw = [cos(a) -sin(a) 0;
           sin(a) cos(a)  0;
           0        0     1];
    pitch = [cos(B)  0 sin(B);
             0       1    0;
             -sin(B) 0 cos(B)];
    roll = [1     0     0;
            0 cos(y) -sin(y);
            0 sin(y) cos(y)];

    RotatedTestLines = yaw*pitch*roll*TestLines.';

    % If any dist is less than zero, than the above centroid calculation
    % method from the library gave us a poor result.
    % (This suggests that one of the lines never intersect the shape, and
```

```
% we cannot calculate a probability distribution from this) Thus,
% recalculate the centroid using a slower, more accurate method.
dist = max(V1 * RotatedTestLines);
if any(dist<=0)
    disp("CENTROID IS REALLY OFF. Using slower centroid calculation method.")
    k = convhulln(V);
    C = centroid(V(unique(k), :));
    trans = createTranslation3d(-C(1), -C(2), -C(3));
    assignin('base', 'V1', V);
    assignin('base', 'C', C);
    assignin('base', 'F', F)

    [V, F] = transformMesh(V, F, trans);
    dist = max(V * RotatedTestLines);
else
    V = V1;
end

assignin('base', 'RotatedTestLines', RotatedTestLines);
assignin('base', 'dist', dist);
assignin('base', 'V', V);

if strcmp(angleChoice, 'cubesup')
    lineIndex = randsample(numLines, 1, true, 1./(dist.^3));
else
    error('angleChoice must be a valid stochastic category')
end

direction = RotatedTestLines(:,lineIndex).';

% max distance associated with the chosen direction
maxD = dist(lineIndex);

if probabilistic
    while true
        % random cut from possible cuts that may cut away delta, assuming
        % convex shape (maxD-maxD*(2delta)^(1/3): maxD)
        d = rand()*maxD*(delta*2)^(1/3)*1.1 + maxD*(1-(delta*2)^(1/3)*1.1);

        plane = createPlane(d*direction, direction);
        [V1, F1] = clipConvexPolyhedronHP(V, F, plane);

        printVolume = meshVolume(V1, F1);

        cutVolume = origVolume - printVolume;

        acceptParam = exprnd(origVolume*delta);

        if acceptParam >= cutVolume
            break
        end
    end
    V = V1;
    F = F1;
    cutRatio = cutVolume/origVolume;
    probabilisticCutRatios = [probabilisticCutRatios cutRatio];
else
```

```
        % determine amount of volume to cut away
        deltaTimesVolume = 1-delta;

        % iterate through all distances possible, find cuts that gives
        % deltaTimesVolume proportion of volume cut away
        for d=maxD*(1-delta):-maxD/1000:0

             plane = createPlane(d*direction, direction);
             [V, F] = clipConvexPolyhedronHP(V, F, plane);

             printVolume = meshVolume(V, F);

             if deltaTimesVolume*origVolume >= printVolume
                 break
             end
        end
    end

    % set the new original volume to be the volume of the cut
    origVolume = printVolume;

    if animated
        if (mod(i,Frames)==0)
            cla
            drawMesh(V, F); view(3); alpha(0.5)
            pause(0.001)
            frame = getframe(gcf, [50 25 460 380]);
            writeVideo(writerObj,frame);
        end
    else
        if (mod(i,Frames)==0)
            norms = vecnorm([ xValues; yValues ]);
            if max(norms) < maxNorm / 200
                disp('broke')
                break
            end
        end
        if (mod(i,100)==0)
            drawMesh(V, F); view(3);
            pause(0.001)
        end
    end

    if probabilistic
        if mod(i, 100) == 0
            disp("After " + i + " iterations, the average ratio of volume cut is: "
            + mean(probabilisticCutRatios));
        end
    end

    if i == 1000
        beep
        toc
        disp("-----REACHED 1000 ITERATIONS-----")
        pause(2)
    end
end
```

```
assignin('base', 'probabilisticCutRatios', probabilisticCutRatios);

% if animated or not, make final adjustments, close writer objects
if ~animated
    hold off
else
    close(writerObj);

    figure;
    drawMesh(V, F); view(3);
    axis equal
    pause(0.001)
end

if warningIdentifier ~= -1
    warning('on', warningIdentifier)
end

beep

end

%%%%%% Examples of curves to evolve via Discretized Shaving of Stones %%%%%%

% Sphere:
% N = 20;
% M = 40;
% r = ones(N+1, M); % all radii for first row and last row must be the same
% phi = 0:pi/N:pi;
% theta = 0:2*pi/M:2*pi-2*pi/M;
% PolygonSlicing3D(r,theta,phi,3000,5,'cubesup',false,true,true);

% Egg-shape:
% N = 20;
% M = 40;
% r = ones(N+1, M); % all radii for first row and last row must be the same
% phi = 0:pi/N:pi;
% theta = 0:2*pi/M:2*pi-2*pi/M;
% r = r .* reshape((cos(pi-phi).^2 + 1.04*cos(pi-phi) + 2.3),[N+1,1]);
% PolygonSlicing3D(r,theta,phi,3000,5,'cubesup',false,true,true);

% Ellipsoid:
% a = 5;
% b = 2;
% c = 9;
% N = 20;
% M = 40;
% r = ones(N+1, M); % all radii for first row and last row must be the same
% phi = 0:pi/N:pi;
% theta = 0:2*pi/M:2*pi-2*pi/M;
% denom = ones(N+1, M);
% denom1 = denom .* (reshape(cos(phi), [N+1,1]).^2 * a^2 * b^2);
% denom2 = denom .* (reshape(cos(theta), [1,M]).^2*b^2 + reshape(sin(theta), [1,M]).^2*a^2);
% denom2 = denom2 .* (reshape(sin(phi), [N+1,1]).^2 * c^2);
% denom = (denom1 + denom2).^(0.5);
% r = r * a*b*c ./denom;
```

```
% PolygonSlicing3D(r,theta,phi,3000,5,'cubesup',false,true,true);

% Tetrahedron:
% [V, ~, F] = createTetrahedron();
% V(1, :) = [-1 -2 -3];
% V(3, :) = [1.5 -7 1.2];
% PolygonSlicing3D(V,F,[],3000,5,'cubesup',false,true,false);

% Inital Shape for Sphere, Egg-Shape, Ellipsoid
% NOTE: First, run the code in the Sphere, Egg-Shape, etc. blocks,
% not including the last line that proceeds with the evolution. This
% produces the spherical coordinates of the shapes. Then, run the following
% code to produce the initial shape:
% % START
% width = size(r,2);
% height = size(r,1);
%
% % Initial Condition
% theta = reshape(theta,[1,width]);
% phi = reshape(phi,[height,1]);
%
% X = (r.*sin(phi)).*cos(theta);
% Y = (r.*sin(phi)).*sin(theta);
% Z = r.*cos(phi);
%
% X = flipud(X);
% Y = flipud(Y);
% Z = flipud(Z);
%
% [V, F] = surfToMesh(X,Y,Z);
%
% C = polyhedronCentroid(V, F);
% X = X - C(1);
% Y = Y - C(2);
% Z = Z - C(3);
%
% [V, F] = surfToMesh(X,Y,Z);
% figure;
% drawMesh(V, F); view(3);
% axis equal
% pause(0.001)

% Initial Shape for Tetrahedron
% NOTE: Run the code in the Tetrahedron block, not including the last line,
% to initialize the vertices. Then, run the following code to plot the
% initial shape:
% % START
% C = polyhedronCentroid(V, F);
%
% trans = createTranslation3d(-C(1), -C(2), -C(3));
%
% [V, F] = transformMesh(V, F, trans);
% figure;
% drawMesh(V, F); view(3);
% axis equal
% pause(0.001)
```

6 Numerical_Solve_Curve_2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%% Numerical_Solve_Curve_2.m %%%%%%%%%%%%%%%%%%%%%%%%%

function [matrix1,matrix2] = Numerical_Solve_Curve_2(gamma01,gamma02,L,M,N,Frames,expNum,useSupport)

% DESCRIPTION: This an implementation of curve shortening flow in the
% plane. We employ a semi-implicit finite difference scheme. This
% method will take a C^1, closed, embedded plane curve and deform it
% according to curve shortening flow for the life of the flow (see Tmax
% below)

% INPUT: (gamma01,gamma02) is an inline parametric plane curve with
% period L. The approximate solution we will generate is defined on
% [0,Tmax] x [0,L]. We will use M subintervals for [0,Tmax] and N
% subintervals for [0,L].

% OUTPUT: A sequence of plots of the approximate solution curves


% "delta p"
k=L/N;


% Intialize gamma matrix: gamma1(i,j) approximates the true solution
% gamma1(ti,pj), where ti=i*h, i=0,...,M and pj=j*k, j=0,...,N, and similar
% for gamma2(i,j)
gamma1=zeros(M*15+1,N+1);
gamma2=zeros(M*15+1,N+1);

% Initial Condition
p=(0:k:L);
gamma1(1,:)=gamma01(p);
gamma2(1,:)=gamma02(p);

% Using polygonal Formula for Tmax
Tmax=.99*polyarea(gamma01(p),gamma02(p))/(2*pi);

% "delta t"
t=Tmax/M;

int = 0:2*pi/N:2*pi;
unitcirc = zeros(N+1, 2);
unitcirc(:, 1) = cos(int);
unitcirc(:, 2) = sin(int);

polyin = polyshape(gamma1(1, 1:N),gamma2(1, 1:N));

[centerOfGravX, centerOfGravY] = centroid(polyin);

gamma1(1, :) = gamma1(1, :) - centerOfGravX;
gamma2(1, :) = gamma2(1, :) - centerOfGravY;

avgRad = sqrt(area(polyin)/pi);

gamma1(1, :) = gamma1(1, :) / avgRad;
gamma2(1, :) = gamma2(1, :) / avgRad;
```

```
% Plot initial condition gamma0
myfig=plot(gamma1(1, :),gamma2(1, :));
pause(0.001)

% Setting the axis of the figure
axis([min(gamma1(1, :)) max(gamma1(1, :)) min(gamma2(1, :)) max(gamma2(1, :))])
axis equal

warningIdentifier = -1;
animated = true;

if animated
    writerObj = VideoWriter('example.avi');
    writerObj.FrameRate = 60;
    open(writerObj);

    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');
else
    hold on
end

maxNorm = max(vecnorm([ gamma1(1,:); gamma2(1,:) ]));

% Semi-Implicit Scheme
for i=1:M*15

    polyin = polyshape(gamma1(i, 1:N),gamma2(i, 1:N));

    if warningIdentifier == -1
        [msgStr, msgId] = lastwarn;
        warningIdentifier = msgId;
        warning('off', msgId)
        warning(msgStr)
    end

    curArea = area(polyin);

    avgRad = sqrt(curArea/pi);

    [centerOfGravX, centerOfGravY] = centroid(polyin);

    gamma1(i, :) = gamma1(i, :) - centerOfGravX;
    gamma2(i, :) = gamma2(i, :) - centerOfGravY;

    gamma1(i, :) = gamma1(i, :) / avgRad;
    gamma2(i, :) = gamma2(i, :) / avgRad;

    dotproducts = unitcirc * [gamma1(i, :); gamma2(i, :)];
    rad = max(dotproducts.');

    if i > 1
        ratio = sqrt(curArea/prevArea) / avgRad;
        if all((abs(gamma1(i, :) - ratio*gamma1(i-1, :))) < 10^-6) && all((abs(gamma2(i, :) -
        ratio*gamma2(i-1, :))) < 10^-6)
            disp("found limiting shapeeeee")
```

```
        break
    end
end

triDiagMat = zeros(N+1);

hSquared = zeros(N+1, 1);

for j=2:N+1

    hSquared(j) = (gamma1(i,j)-gamma1(i,j-1))^2 + (gamma2(i,j)-gamma2(i,j-1))^2 + 10^(-16);

end

hSquared(1) = hSquared(N+1);

for j=1:N

    if expNum == 0
        distanceDenom = 1;
    else
        if useSupport
            distanceDenom = ((rad(j)/avgRad)^expNum);
        else
            distanceDenom = ((norm([(gamma1(i, j) - centerOfGravX) (gamma2(i, j) -
            centerOfGravY)])/avgRad)^expNum);
        end
    end

    prev = j-1;
    if prev == 0
        prev = N;
    end

    triDiagMat(j, prev) = -1 / distanceDenom;
    triDiagMat(j, j+1) = -1 / distanceDenom;
    triDiagMat(j, j) = 1/(2*t)*(hSquared(j) + hSquared(j+1)) + 2 / distanceDenom;

end

triDiagMat(N+1, N) = triDiagMat(1, N);
triDiagMat(N+1, 2) = triDiagMat(1, 2);
triDiagMat(N+1, N+1) = triDiagMat(1, 1);

vectorComplex = zeros(N+1, 1);

for j=1:N

    vectorComplex(j) = 1/(2*t)*complex(gamma1(i, j), gamma2(i, j)) * (hSquared(j) + hSquared(j+1));

end

vectorComplex(N+1) = vectorComplex(1);

newVecComplex = triDiagMat\vectorComplex;

gamma1(i+1, :) = real(newVecComplex);
```

```
    gamma2(i+1, :) = imag(newVecComplex);

    if (mod(i,Frames)==0)
        disp(max(gamma2(i+1, :))/max(gamma1(i+1, :)))
        pause(0.001)
        set(myfig,'XData', gamma1(i+1,:))
        set(myfig,'YData', gamma2(i+1,:))
    end
    prevArea = curArea;
end

matrix1 = gamma1;
matrix2 = gamma2;

end

%%%%%% Examples of smooth embedded plan curves to evolve via CSF ?? %%%%%%

% Unit circle: x^2+y^2=1
% [m1, m2] = Numerical_Solve_Curve_2(@cos,@sin,2*pi,20000,100,100,3,true);

% Ellipse: (x/3)^2+(y/2)^2=1
% [m1, m2] = Numerical_Solve_Curve_2(@(p)2*cos(p),@(p)3*sin(p),2*pi,15000,200,120,3,true);

% Ellipse, different form: x^2+(y/b)^2=1, b = 0.9
% [m1, m2] = Numerical_Solve_Curve_2(@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*cos(p),
@(p)0.9./(0.81*cos(p).^2 + sin(p).^2).^(1/2).*sin(p),2*pi,15000,200,120,3,true);
% [m1, m2] = Numerical_Solve_Curve_2(@(p)0.7./(0.49*cos(p).^2 + sin(p).^2).^(1/2).*cos(p),
@(p)0.7./(0.49*cos(p).^2 + sin(p).^2).^(1/2).*sin(p),2*pi,15000,200,120,3,true);

% Egg-shape: k(cos^2(theta) + a*cos(theta) + b), b = 0.9
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*cos(p),
@(p)(cos(p).^2 + 1.04*cos(p) + 2.3).*sin(p),2*pi,15000,200,120,3,true);

% Triangle ISOC
% [m1, m2] = Numerical_Solve_Curve_2(@(p)radiusisocx(p), @(p)radiusisocy(p),2*pi,15000,200,120,3,true);

% Circle dent
% [m1, m2] = Numerical_Solve_Curve_2(@(p)circledentx(p), @(p)sin(p),2*pi,15000,200,50,3,true);

% Non-embedded curve:
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(cos(2*p).*cos(p)),
@(p)(cos(2*p).*sin(p)),2*pi,15000,200,120,3,true);

% Folium:
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(-(cos(p).^3).*cos(p)),
@(p)(-(cos(p).^3).*sin(p)),2*pi,15000,200,100,3,true);
% Row 7598

% Limcon of Pascal: r = 3/2 + cos(theta)
% Numerical_Solve_Curve_2(@(p)((3/2 + cos(p)).*cos(p)),
@(p)((3/2 + cos(p)).*sin(p)),2*pi,15000,200,100,3,true);

% f = @() Numerical_Solve_Curve_2(@(p)((3/2 + cos(p)).*cos(p)),@(p)((3/2 + cos(p)).*sin(p)),2*pi,15000,20
% timeit(f)

% Lame curve: (x/3)^1/4+(y/4)^1/4=1
```

```
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(abs(cos(p)).^(1/2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(1/2) * 4 .* sign(sin(p))),2*pi,15000,200,100,3,true);

% Another Lame curve:
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(abs(cos(p)).^(2) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(2) * 4 .* sign(sin(p))),2*pi,15000,200,100,3,true);

% Polygon:
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* cos(p)),
@(p)(cos(pi/8)./cos(p - pi/4*floor((8*p + pi)/(2*pi))) .* sin(p)),2*pi,15000,200,100,3,true);

% Non-embedded Lame curve:
% [m1, m2] = Numerical_Solve_Curve_2(@(p)(abs(cos(p)).^(4) * 3 .* sign(cos(p))),
@(p)(abs(sin(p)).^(4) * 4 .* sign(sin(p))),2*pi,15000,200,100,3,true);


% Triangle ISOC (ADD SEMICOILOOOOOOONNNNNNNNNNNNNNN)
% [m1, m2] = Numerical_Solve_Curve_2(@(p)trianglex(p), @(p)triangley(p),2*pi,15000,200,50,3,true);
```