**Ten Test Cases for Frechet Distance**

An algorithm to compute Frechet distance was added to the existing Frechet decision problem script by computing the critical distances. It was tested on randomly generated polygonal curves. For P and Q below, the first row is the x-coordinate and the second is the y-coordinate.

P =

4 28 15 47 11 26 3 27 16 41

34 19 34 25 3 12 30 22 16 12

Q =

6 4 24 33 16 12 5 39

36 36 36 22 22 5 19 13

 

P =

10 31 38 7 37 14 50

41 10 3 3 25 0 13

Q =

10 13 27 35 12 27 31 37 24 52

40 43 36 5 20 4 1 31 35 14

 

P =

7 24 13 27 26 16 45 15 47 25

5 15 18 36 38 6 41 44 0 4

Q =

5 7 7 33 40 31 25 24

6 6 25 23 27 27 29 4

 

P =

3 9 17 20 49

45 19 0 1 9

Q =

4 10 38 11 8 50

44 47 33 21 21 8

 

P =

27 36 41 46 50 46

15 26 21 38 11 44

Q =

25 15 5 46 1 46

17 26 43 30 40 42

 

P =

38 40 24 19 32 13 9 18 19 48

37 36 2 26 4 37 39 18 28 18

Q =

36 4 21 28 47 30 1 23 46

36 19 41 38 5 10 9 21 17

 

after the first pass: 24.396072 < len <= 24.758837

in the 2nd pass: len = 24.521894

P =

21 28 46 22 30 36 27 17

22 0 46 42 24 38 25 16

Q =

19 1 47 40 15

23 39 31 45 14

 

after the first pass: 29.529646 < len <= 29.832868

frechet distance not found in the 2nd pass: len = 29.832868

P =

10 35 42 6 50 9

35 39 37 3 25 22

Q =

12 24 21 49 31 30 7 48 10

35 21 38 39 20 8 35 6 24

 

after the first pass: 25.238859 < len <= 25.447355

frechet distance not found in the 2nd pass: len = 25.447355

P =

33 34 27 0 28 14 15 31 4 28

19 1 11 24 14 27 47 36 18 13

Q =

33 43 34 31 43 41 28

21 48 0 44 12 38 15

 

after the first pass: 31.144823 < len <= 31.468318

in the 2nd pass: len = 31.184332

P =

35 31 0 26 9 18 20 15 26

10 24 39 10 14 25 40 34 7

Q =

33 42 9 42 47 34 37 10 0 26

10 44 11 44 33 15 21 11 30 8

 

after the first pass: 22.825424 < len <= 23.537205

in the 2nd pass: len = 23.447415

# Matlab Script

## Main

%-- Script to run the Decision Problem (DP) for Frechet distance

%-- The DP is based on Algorithm 1 in "Computing the Frechet Distance

%-- Between Two Polygonal Curves", Alt and Godau, International Journal of

%-- Computational Geometry and Applications, 1995.

%

%-- Author: Rich Kenefic

%-- Created: 14-Dec-2011

%-- Modified: 27-Apr-2012

% Branched to RevA to capture intermediate results for use in

% frechet\_decide and frechet\_compute.

%-- Added other edge cases into the type A critical points

% missed by Alt and Godau

%

clear all

close all

global I J lP lQ lPQ bP bQ

choice = menu('option','Std.','New Random','Re-Run Last Random');

switch choice

case 1 %--Standard Instance

S1 = RandStream('swb2712','seed',0); %--default initial state

S2 = RandStream('swb2712','seed',1); %--used for quicksort

case 2

clk = sum(100\*clock);

S1 = RandStream('swb2712','seed',clk); %--save state

S2 = RandStream('swb2712','seed',clk+1);

save FRECHET\_starting\_clk.mat clk

fprintf('\*\*\*\*\*initialization of rand\*\*\*\*\*\n');

fprintf(' clk = %d\n',clk);

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n');

case 3

load FRECHET\_starting\_clk.mat clk

S1 = RandStream('swb2712','seed',clk); %--use last state

S2 = RandStream('swb2712','seed',clk+1);

end

RandStream.setDefaultStream(S1);

%--no need to identify the stream for rand or other calls, the stream

%--ID is only needed for quicksort (later, in the optimization problem)

%-- Create two Polygonal curves, P (man's path) and Q (dog's path)

%-- where P(1,i) and P(2,i) are the (x,y) coordinates of the

%-- ith vertex in P, Q(1,j) and Q(2,j) are the (x,y) coordinates

%-- of the jth vertex in Q

I = randi([5 10]); %--number of vertices in P

J = randi([5 10]); %--number of vertices in Q

for i=1:I

P(1,i) = randi([0 50]); P(2,i) = randi([0 50]);

end

%--keep the start and stop for P and Q close

Q(1,1) = P(1,1) + randi([-2 2]); Q(2,1) = P(2,1) + randi([-2 2]);

for j=2:J-1

Q(1,j) = randi([0 50]); Q(2,j) = randi([0 50]);

end

Q(1,J) = P(1,I) + randi([-2 2]); Q(2,J) = P(2,I) + randi([-2 2]);

figure(1)

plot(P(1,:),P(2,:),'g-',Q(1,:),Q(2,:),'r:'); hold on;

plot([P(1,1) P(1,1)],[P(2,1) P(2,1)],'go');

plot([P(1,I) P(1,I)],[P(2,I) P(2,I)],'gx');

plot([Q(1,1) Q(1,1)],[Q(2,1) Q(2,1)],'ro');

plot([Q(1,J) Q(1,J)],[Q(2,J) Q(2,J)],'rx');

%--initialize data structures

frechet\_init(P,Q);

%-- request a leash length

len = input('Enter the leash length\n');

while ~isempty(len)

%--decide if the leash is long enough

decision = frechet\_decide\_RevA(P,Q,len,1,0);

l\_len = len;

if decision==0

fprintf('The leash is too short\n');

else

fprintf('The leash is long enough\n');

end

len = input('Enter the leash length\n');

end

sol\_len = frechet\_compute(P,Q);

figure(1); legend({'man' 'dog' 'start' 'stop' 'start' 'stop'}); axis equal;

title(['Computed length=' num2str(sol\_len) ', last guess=' num2str(l\_len)]);

## Functions

function frechet\_init(P,Q)

%--function to initialize data structures for frechet\_decide and

%--frechet\_compute

global I J lP lQ lPQ bP bQ

[M,N]=size(P);

if M~=2, error('P must be a 2 by I array'); end

if (I ~= N), error('P must be a 2 by I array'); end

[M,N]=size(Q);

if M~=2, error('Q must be a 2 by J array'); end

if (J ~= N), error('Q must be a 2 by J array'); end

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

lP(i) = (xp1-xp)^2 + (yp1-yp)^2; %--length^2 of the ith segment of P

end

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

lQ(j) = (xq1-xq)^2 + (yq1-yq)^2; %--length^2 of the jth segment of Q

end

for i=1:I

xp = P(1,i); yp = P(2,i);

for j=1:J

xq = Q(1,j); yq = Q(2,j);

lPQ(i,j) = (xq-xp)^2 + (yq-yp)^2; %--length^2 of P->Q link

end

end

%--Interior

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

bP(i,j) = 2\*((xp-xq)\*(xp1-xp)+(yp-yq)\*(yp1-yp));

bQ(i,j) = 2\*((xq-xp)\*(xq1-xq)+(yq-yp)\*(yq1-yq));

end

end

%--Top row

xp = P(1,I); yp = P(2,I);

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

bQ(I,j) = 2\*((xq-xp)\*(xq1-xq)+(yq-yp)\*(yq1-yq));

end

%--Right column

xq = Q(1,J); yq = Q(2,J);

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

bP(i,J) = 2\*((xp-xq)\*(xp1-xp)+(yp-yq)\*(yp1-yp));

end

%--Top Right cell

xp = P(1,I-1); yp = P(2,I-1);

xp1 = P(1,I); yp1 = P(2,I);

xq = Q(1,J-1); yq = Q(2,J-1);

xq1 = Q(1,J); yq1 = Q(2,J);

bP(I,J) = 2\*((xp-xq1)\*(xp1-xp)+(yp-yq1)\*(yp1-yp));

bQ(I,J) = 2\*((xq-xp1)\*(xq1-xq)+(yq-yp1)\*(yq1-yq));

return

function [decide] = frechet\_decide\_RevA(P,Q,len,plotFSD,printFSD)

%--solves the decision problem for Frechet distance

%--modified: 27 Apr 2012: To compute data required in frechet\_init

global I J lP lQ lPQ bP bQ

[M,N]=size(P);

if M~=2, error('P must be a 2 by I array'); end

if (I ~= N), error('P must be a 2 by I array'); end

[M,N]=size(Q);

if M~=2, error('Q must be a 2 by J array'); end

if (J ~= N), error('Q must be a 2 by J array'); end

%--compute the free space in each cell

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

%--solve for the line segment circle intersections

ap = lP(i); aq = lQ(j);

bp = bP(i,j);

bq = bQ(i,j);

c = lPQ(i,j) - len^2;

dp = bp\*bp - 4\*ap\*c;

dq = bq\*bq - 4\*aq\*c;

if dp<0.0, %-- a\_ij, b\_ij, LF\_ij

%--line and circle do not intersect

A(i,j) = NaN; B(i,j) = NaN;

LF(i,j,1) = NaN; LF(i,j,2) = NaN;

else

up = (-bp+sqrt(dp))/(2\*ap); um = (-bp-sqrt(dp))/(2\*ap);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

A(i,j) = NaN; B(i,j) = NaN;

LF(i,j,1) = NaN; LF(i,j,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

A(i,j) = 0; B(i,j) = 1;

LF(i,j,1) = 0; LF(i,j,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (b\_i,j)

A(i,j) = 0; B(i,j) = max([um up]);

LF(i,j,1) = 0; LF(i,j,2) = B(i,j);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

A(i,j) = min([um up]); B(i,j) = 1;

LF(i,j,1) = A(i,j); LF(i,j,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

A(i,j) = min([um up]); B(i,j) = max([um up]);

LF(i,j,1) = A(i,j); LF(i,j,2) = B(i,j);

else

error('Unexpected case in frechet\_decide at LF');

end

end

if dq<0.0, %-- c\_ij, d\_ij, BF\_ij

%--line and circle do not intersect

C(i,j) = NaN; D(i,j) = NaN;

BF(i,j,1) = NaN; BF(i,j,2) = NaN;

else

up = (-bq+sqrt(dq))/(2\*aq); um = (-bq-sqrt(dq))/(2\*aq);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

C(i,j) = NaN; D(i,j) = NaN;

BF(i,j,1) = NaN; BF(i,j,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

C(i,j) = 0; D(i,j) = 1;

BF(i,j,1) = 0; BF(i,j,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (d\_i,j)

C(i,j) = 0; D(i,j) = max([um up]);

BF(i,j,1) = 0; BF(i,j,2) = D(i,j);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

C(i,j) = min([um up]); D(i,j) = 1;

BF(i,j,1) = C(i,j); BF(i,j,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

C(i,j) = min([um up]); D(i,j) = max([um up]);

BF(i,j,1) = C(i,j); BF(i,j,2) = D(i,j);

else

error('Unexpected case in frechet\_decide at BF');

end

end

end

end

%--Top row

xp = P(1,I); yp = P(2,I);

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

%--solve for the line segment circle intersections

aq = lQ(j);

bq = bQ(I,j);

c = lPQ(I,j) - len^2;

dq = bq\*bq - 4\*aq\*c;

if dq<0.0, %-- c\_ij, d\_ij, BF\_ij

%--line and circle do not intersect

C(I,j) = NaN; D(I,j) = NaN;

BF(I,j,1) = NaN; BF(I,j,2) = NaN;

else

up = (-bq+sqrt(dq))/(2\*aq); um = (-bq-sqrt(dq))/(2\*aq);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

C(I,j) = NaN; D(I,j) = NaN;

BF(I,j,1) = NaN; BF(I,j,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

C(I,j) = 0; D(I,j) = 1;

BF(I,j,1) = 0; BF(I,j,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (d\_i,j)

C(I,j) = 0; D(I,j) = max([um up]);

BF(I,j,1) = 0; BF(I,j,2) = D(I,j);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

C(I,j) = min([um up]); D(I,j) = 1;

BF(I,j,1) = C(I,j); BF(I,j,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

C(I,j) = min([um up]); D(I,j) = max([um up]);

BF(I,j,1) = C(I,j); BF(I,j,2) = D(I,j);

else

error('Unexpected case in frechet\_decide at BF (top row)');

end

end

end

%--Right column

xq = Q(1,J); yq = Q(2,J);

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

%--solve for the line segment circle intersections

ap = lP(i);

bp = bP(i,J);

c = lPQ(i,J) - len^2;

dp = bp\*bp - 4\*ap\*c;

if dp<0.0, %-- a\_ij, b\_ij, LF\_ij

%--line and circle do not intersect

A(i,J) = NaN; B(i,J) = NaN;

LF(i,J,1) = NaN; LF(i,J,2) = NaN;

else

up = (-bp+sqrt(dp))/(2\*ap); um = (-bp-sqrt(dp))/(2\*ap);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

A(i,J) = NaN; B(i,J) = NaN;

LF(i,J,1) = NaN; LF(i,J,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

A(i,J) = 0; B(i,J) = 1;

LF(i,J,1) = 0; LF(i,J,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (b\_i,j)

A(i,J) = 0; B(i,J) = max([um up]);

LF(i,J,1) = 0; LF(i,J,2) = B(i,J);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

A(i,J) = min([um up]); B(i,J) = 1;

LF(i,J,1) = A(i,J); LF(i,J,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

A(i,J) = min([um up]); B(i,J) = max([um up]);

LF(i,J,1) = A(i,J); LF(i,J,2) = B(i,J);

else

error('Unexpected case in frechet\_decide at LF (right col)');

end

end

end

%--Top Right cell

xp = P(1,I-1); yp = P(2,I-1);

xp1 = P(1,I); yp1 = P(2,I);

xq = Q(1,J-1); yq = Q(2,J-1);

xq1 = Q(1,J); yq1 = Q(2,J);

ap = lP(I-1); aq = lQ(J-1);

bp = bP(I,J);

bq = bQ(I,J);

cp = lPQ(I-1,J) - len^2;

cq = lPQ(I,J-1) - len^2;

dp = bp\*bp - 4\*ap\*cp;

dq = bq\*bq - 4\*aq\*cq;

if dp<0.0, %-- a\_ij, b\_ij, LF\_ij

%--line and circle do not intersect

A(I,J) = NaN; B(I,J) = NaN;

LF(I,J,1) = NaN; LF(I,J,2) = NaN;

else

up = (-bp+sqrt(dp))/(2\*ap); um = (-bp-sqrt(dp))/(2\*ap);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

A(I,J) = NaN; B(I,J) = NaN;

LF(I,J,1) = NaN; LF(I,J,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

A(I,J) = 0; B(I,J) = 1;

LF(I,J,1) = 0; LF(I,J,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (b\_i,j)

A(I,J) = 0; B(I,J) = max([um up]);

LF(I,J,1) = 0; LF(I,J,2) = B(i,j);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

A(I,J) = min([um up]); B(I,J) = 1;

LF(I,J,1) = A(i,j); LF(I,J,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

A(I,J) = min([um up]); B(I,J) = max([um up]);

LF(I,J,1) = A(I,J); LF(I,J,2) = B(I,J);

else

error('Unexpected case in frechet\_decide at LF (top right cell)');

end

end

if dq<0.0, %-- c\_ij, d\_ij, BF\_ij

%--line and circle do not intersect

C(I,J) = NaN; D(I,J) = NaN;

BF(I,J,1) = NaN; BF(I,J,2) = NaN;

else

up = (-bq+sqrt(dq))/(2\*aq); um = (-bq-sqrt(dq))/(2\*aq);

if (((up<0)&&(um<0))||((up>1)&&(um>1)))

%--line segment outside circle

C(I,J) = NaN; D(I,J) = NaN;

BF(I,J,1) = NaN; BF(I,J,2) = NaN;

elseif ((min([um up])<0)&&(max([um up])>1))

%--line segment is interior to circle

C(I,J) = 0; D(I,J) = 1;

BF(I,J,1) = 0; BF(I,J,2) = 1;

elseif ((min([um up])<=0)&&(max([um up])<=1))

%--one intersection (d\_i,j)

C(I,J) = 0; D(I,J) = max([um up]);

BF(I,J,1) = 0; BF(I,J,2) = D(I,J);

elseif ((min([um up])>=0)&&(max([um up])>1))

%--one intersection (a\_i,j)

C(I,J) = min([um up]); D(I,J) = 1;

BF(I,J,1) = C(I,J); BF(I,J,2) = 1;

elseif ((min([um up])>=0)&&(max([um up])<=1))

%--two intersections

C(I,J) = min([um up]); D(I,J) = max([um up]);

BF(I,J,1) = C(I,J); BF(I,J,2) = D(I,J);

else

error('Unexpected case in frechet\_decide at BF (top right cell)');

end

end

if plotFSD==1 %--plot the free space diagram

figure(2)

if exist('h')==1

close(2)

figure(2)

h = [];

end

ih = 1;

for i=1:I-1

for j=1:J-1

x = [j-1 j j j-1]; y = [i-1 i-1 i i];

h(ih)=patch(x',y','k'); ih = ih + 1;

x = []; y = [];

if ~isnan(C(i,j))

x = [x j-1+C(i,j)]; y = [y i-1];

end

if ~isnan(D(i,j))

x = [x j-1+D(i,j)]; y = [y i-1];

end

if ~isnan(A(i,j+1))

x = [x j]; y = [y i-1+A(i,j+1)];

end

if ~isnan(B(i,j+1))

x = [x j]; y = [y i-1+B(i,j+1)];

end

if ~isnan(D(i+1,j))

x = [x j-1+D(i+1,j)]; y = [y i];

end

if ~isnan(C(i+1,j))

x = [x j-1+C(i+1,j)]; y = [y i];

end

if ~isnan(B(i,j))

x = [x j-1]; y = [y i-1+B(i,j)];

end

if ~isnan(A(i,j))

x = [x j-1]; y = [y i-1+A(i,j)];

end

if length(x)>2

h(ih) = patch(x',y','w'); ih = ih + 1;

elseif length(x)>0

fprintf('patch error\n');

end

end

end

end

%--Compute the reachable sets for each cell

for i=2:I %--fill in column 1

LR(i,1,1) = NaN; LR(i,1,2) = NaN;

end

for j=2:J %--fill in row 1

BR(1,j,1) = NaN; BR(1,j,2) = NaN;

end

for i=1:I-1

for j=1:J-1

if (i==1)&&(j==1)

if (LF(i,j,1)==0)&&(BF(i,j,1)==0) %--start at the origin

LR(i,j+1,1) = LF(i,j+1,1); LR(i,j+1,2) = LF(i,j+1,2);

BR(j+1,i,1) = BF(j+1,i,1); BR(j+1,i,2) = BF(j+1,i,2);

else

LR(i,j+1,1) = NaN; LR(i,j+1,2) = NaN;

BR(j+1,i,1) = NaN; BR(j+1,i,2) = NaN;

end

else

if isnan(LR(i,j,1))&&isnan(BR(i,j,1))

LR(i,j+1,1) = NaN; LR(i,j+1,2) = NaN;

BR(i+1,j,1) = NaN; BR(i+1,j,2) = NaN;

elseif (~isnan(LR(i,j,1)))&&isnan(BR(i,j,1))

BR(i+1,j,1) = BF(i+1,j,1); BR(i+1,j,2) = BF(i+1,j,2);

if (LF(i,j+1,2)<LR(i,j,1))||isnan(LF(i,j+1,2))

LR(i,j+1,1) = NaN; LR(i,j+1,2) = NaN;

elseif LF(i,j+1,1)>LR(i,j,1)

LR(i,j+1,1) = LF(i,j+1,1);

LR(i,j+1,2) = LF(i,j+1,2);

else

LR(i,j+1,1) = LR(i,j,1);

LR(i,j+1,2) = LF(i,j+1,2);

end

elseif isnan(LR(i,j,1))&&(~isnan(BR(i,j,1)))

LR(i,j+1,1) = LF(i,j+1,1); LR(i,j+1,2) = LF(i,j+1,2);

if (BF(i+1,j,2)<BR(i,j,1))||isnan(BF(i+1,j,2))

BR(i+1,j,1) = NaN; BR(i+1,j,2) = NaN;

elseif BF(i+1,j,1)>BR(i,j,1)

BR(i+1,j,1) = BF(i+1,j,1);

BR(i+1,j,2) = BF(i+1,j,2);

else

BR(i+1,j,1) = BR(i,j,1);

BR(i+1,j,2) = BF(i+1,j,2);

end

else

LR(i,j+1,1) = LF(i,j+1,1); LR(i,j+1,2) = LF(i,j+1,2);

BR(i+1,j,1) = BF(i+1,j,1); BR(i+1,j,2) = BF(i+1,j,2);

end

end

if printFSD==1

fprintf('cell(%d,%d):\n',i,j);

fprintf('\tLF(i,j)=[%f,%f] ',LF(i,j,1),LF(i,j,2));

fprintf('\tBF(i,j)=[%f,%f] \n',BF(i,j,1),BF(i,j,2));

fprintf('\tLR(i,j)=[%f,%f] ',LR(i,j,1),LR(i,j,2));

fprintf('\tBR(i,j)=[%f,%f] \n',BR(i,j,1),BR(i,j,2));

fprintf('\tLR(i,j+1)=[%f,%f] ',LR(i,j+1,1),LR(i,j+1,2));

fprintf('\tBR(i+1,j)=[%f,%f] \n',BR(i+1,j,1),BR(i+1,j,2));

end

end

end

%--decide

if (BR(I,J-1,2)==1)||(LR(I-1,J,2)==1)

decide = 1;

else

decide = 0;

end

if plotFSD==1, title(['Free Space for Leash Length = ' num2str(len)]); end

return

end

function sol\_len = frechet\_compute(P,Q)

%-- function to compute the frechet distance between two polygonal curves

%--modified: 1-May-2012 To include other edge cases in type A missed in

% Alt and Godau

global I J lP lQ lPQ bP bQ

[M,N]=size(P);

if M~=2, error('P must be a 2 by I array'); end

if (I ~= N), error('P must be a 2 by I array'); end

[M,N]=size(Q);

if M~=2, error('Q must be a 2 by J array'); end

if (J ~= N), error('Q must be a 2 by J array'); end

%--type A critical values

%E1(1) = sqrt(lPQ(1,1)); %--starting length

%E1(2) = sqrt(lPQ(I,J)); %--ending length

E1 = sqrt(reshape(lPQ,1,I\*J));

%ecount = 2;

ecount = I\*J;

%--type B critical values

for i = 1:I-1

ap = lP(i);

for j=1:J-1

aq = lQ(j);

bp = bP(i,j);

tst = -bp/(2\*ap);

if (tst>=0)&(tst<=1)

ecount = ecount + 1;

E1(ecount) = sqrt(-((.5\*bp)^2/ap)+lPQ(i,j));

end

bq = bQ(i,j);

tst = -bq/(2\*aq);

if (tst>=0)&(tst<=1)

ecount = ecount + 1;

E1(ecount) = sqrt(-((.5\*bq)^2/aq)+lPQ(i,j));

end

end

end

%--First Pass:

%--sort critical values and test using frechet\_decide to bound the length,

%--the result is a pseudometric (satisfies the triangle inequality), but

%--not yet a true metric (see Alt and Godau)

[sortedE indx] = sort(E1);

last\_len = sortedE(1);

last\_decide = frechet\_decide\_RevA(P,Q,last\_len,0,0);

if last\_decide==0

%--the shortest critical distance is not long enough, look for a longer

%--one

LsortedE = length(sortedE);

for i=2:LsortedE

len = sortedE(i);

decide = frechet\_decide\_RevA(P,Q,len,0,0);

if decide==1

%--the frechet distance has been bracketed

min\_len = last\_len;

max\_len = len;

break;

else

last\_len = len;

end

end

%--check the longest critical distance (no break encountered)

if decide==0

%--the longest critical distance is not long enough

min\_len = len;

max\_len = Inf;

end

else

%--the shortest critical distance is long enough

max\_len = last\_len;

min\_len = 0;

end

fprintf('after the first pass: %f < len <= %f\n',min\_len,max\_len);

if max\_len<Inf

soln\_len = max\_len;

else

sol\_len = NaN;

end

%--Second Pass:

%--look for critical values that open up a monotone increasing path from

%--start to end. This is brute force and not efficient (see Alt and Godau).

ecount = 0; E2 = []; sortedE = [];

for i=1:I-1 %--looking for a\_i,j = b\_i,k for some k>j (see Alt and Godau)

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

for j=1:J-1

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

for k=j+1:J

den = bP(i,j)-bP(i,k);

if den~=0

u1 = (lPQ(i,k)-lPQ(i,j))/den;

if (u1>=0)&(u1<=1)

xu = xp + u1\*(xp1-xp); yu = yp + u1\*(yp1-yp);

tst = sqrt((xu-xq)^2 + (yu-yq)^2);

if (tst>min\_len)&(tst<max\_len)

ecount = ecount + 1;

E2(ecount) = tst;

end

end

end

end

end

end

for j=1:J-1 %--looking for c\_i,j = d\_k,j for some k>i (see Alt and Godau)

xq = Q(1,j); yq = Q(2,j);

xq1 = Q(1,j+1); yq1 = Q(2,j+1);

for i=1:I-1

xp = P(1,i); yp = P(2,i);

xp1 = P(1,i+1); yp1 = P(2,i+1);

for k=i+1:I

den = bQ(i,j)-bQ(k,j);

if den~=0

u2 = (lPQ(k,j)-lPQ(i,j))/den;

if (u2>=0)&(u2<=1)

xu = xq + u2\*(xq1-xq); yu = yq + u2\*(yq1-yq);

tst = sqrt((xu-xp)^2 + (yu-yp)^2);

if (tst>min\_len)&(tst<max\_len)

ecount = ecount + 1;

E2(ecount) = tst;

end

end

end

end

end

end

%-- sort the results: the shortest one that passes frechet\_decide is the

%-- frechet distance

if ~isempty(E2)

[sortedE indx] = sort(E2);

LsortedE = length(sortedE);

for i=1:LsortedE

len = sortedE(i);

decide = frechet\_decide\_RevA(P,Q,len,0,0);

if decide==1

%--the frechet distance has been found

sol\_len = len;

fprintf('in the 2nd pass: len = %f \n',sol\_len);

return

end

end

%--frechet not found in the list if no return

end

%--frechet not found in the list: length from the first pass is it

sol\_len = max\_len;

fprintf('frechet distance not found in the 2nd pass: len = %f \n',sol\_len);

return