Non-Euclidean Matlab Functions

23 June 2023

Tom Henderson

University of Utah

NE_already_inverted_pent

```
function b =
NE_already_inverted_pent(pent,inverted)
% NE_already_inverted_pent - check if pentagon
already inverted
% On input:
%      pent (5x2 array): vertexes of a pentagon
%      inverted (nx10 array): linear form of already
inverted pentagons
% On output:
%      b (Boolean): 1 if already inverted; else 0
% Call:
%      if NE_already_inverted_pent(pp,inv)
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%
```

NE_already_inverted_tri

```
function b = NE_already_inverted_tri(T,inverted)
% NE_already_inverted_tri - check if triangle
already inverted
% On input:
%      T (3x2 array): triangle vertexes
```

```
%     inverted (nx6 array): linearized existing
triangles
% On output:
%     b (Boolean): 1 if alreday inverted; else 0
% Call:
%     if NE_already_inverted_tri(T,inverted)
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_angle_2circles

```
function theta = NE_angle_2circles(C1,C2)
% NE_angle_2circles - angle between tangents of two
PD circles
% On input:
%     C1 (1x5 vector): center, radius, pt on circle
1
%     C2 (1x5 vector): center, radius, pt on circle
2
% On output:
%     theta (float): angle between tangents of
circle intersection
% Call:
%     theta = NE_angle_2circles(C1,C2);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_cross_ratio

```
function c = NE_cross_ratio(z1,z2,z3,z4)
```

```
% NE_cross_ratio - compute the cross ratio of 4
complex numbers
% On input:
%      z1 (complex): complex number
%      z2 (complex): complex number
%      z3 (complex): complex number
%      z4 (complex): complex number
% On output:
%      c (complex): cross ratio of z1,z2,z3,z4 (real
if pts on geodesic)
% Call:
%      c = NE_cross_ratio(-sqrt(2),-
1+i,1+i,sqrt(2));
%        == (sqrt(2)+1)/(sqrt(2)-1)  see p. 183
(Stahl)
%        == 5.8284   % Note ln is not applied since
we want the cross ratio
%                      not the distance between z2
and z3
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%
```

NE_D2H

```
function Q = NE_D2H(P)
% NE_D2H - convert PD point to PH
% On input:
%      P (1x2 vector): PD point
% On output:
%      Q (1x2 vector): PH point
% Call:
%      q = NE_D2H([0,1]);
% Author:
%      T. Henderson
```

```
%      UU
%      Summer 2023
%
```

NE_dist_PD_pt_pt

```
function d = NE_dist_PD_pt_pt(pt1,pt2)
% NE_dist_PD_pt_pt - distance btween two PD points
%      pt1 (1x2 vector): x,y values of point 1
%      pt2 (1x2 vector): x,y values of point 2
% On output:
%      d (float): distance between points
% Call:
%      d = NE_dist_PD_pt_pt([0,0.9],[0.5,0.5]);
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%

d = acosh(1+2*(norm(pt2-pt1)^2/((1-norm(pt1)^2)*(1-
norm(pt2)^2))));
```

NE_dist_PH_pt_pt

```
function d = NE_dist_PH_pt_pt(pt1,pt2)
% NE_dist_PH_pt_pt - distance btween two PH points
%      pt1 (1x2 vector): x,y values of point 1
%      pt2 (1x2 vector): x,y values of point 2
% On output:
%      d (float): distance between points
% Call:
%      d = NE_dist_PH_pt_pt([0,0.9],[0.5,0.5]);
% Author:
%      T. Henderson
%      UU
```

```
%     Summer 2023
%
```

NE_dist_pt_line

```
function [d,closest_pt] =
NE_dist_pt_line(point,line)
% NE_dist_pt_line - distance between a point and a
line
% Call: d = NE_dist_pt_line(point,line);
% On input:
%     point: 3D point
%     line:  2x3 matrix of 2 points
% On output:
%     d: Euclidean distance between point and line
% Author:
%     Tom Henderson
%     7 January 2000/modified for NE Summer 2023
% Custom Functions Used:
%     none
% Method:
%     find point on line so that distance to point
is minimized
%
```

NE_equi45

```
function T = NE_equi45
% NE_equi45 - create equilateral 45-degree triangle
%       p. 107 Stahl: find length of side of
equilateral triangle with
%                     pi/4 angle:
%       from p. 105:
%                    cos(beta)cos(gamma) +
cos(alpha)
```

```
%          cosh(a) = ---------------------------------
---
%                           sin(beta)sin(gamma)
%          cosh(a) = (1/2 + sqrt(2)/2)/(1/2) = 1 +
sqrt(2)
%               a  = acosh(") = 1.528
%        Point selection:
%           Point 1: origin: [0,0]
%           Point 2: point along x axis at distance
1.528: [0.64343,0]
%           Point 3: point along 45-degree line
through origin that is
%                    at distance 1.528:
[0.455,0.455]
% On input:
%     N/A
% On output:
%     T (3x2 array): vertexes of triangle
% Call:
%     T = NE_equi45;
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_gen_pentagon

```
function poly = NE_gen_pentagon(b)
% NE_gen_pentagon - generate a tiling pentagon
%     Stahl p.86 Figure 6.7
% On input:
%     b (Boolean): 1: draw pentagon in new figure;
else don't
% On output:
%     poly(poly struct): polygon info
```

```
%        (k).pts (n_k x 2 array): points in k_th
segment
%           .all_pts (nx2 array): all segment points
[only in first segment
%           .corners1 (1x10 vector): corners as
vector: [x1,y1,...,x5,y5]
%           .corners2 (5x2 array): corners as 2D
array
% Call:
%      poly = NE_gen_pentagon(1);
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%
```

NE_H2D

```
function Q = NE_H2D(P)
% NE_H2D - convert PH point to PD
% On input:
%      P (1x2 vector): PH point
% On output:
%      Q (1x2 vector): PD point
% Call:
%      q = NE_H2D([0,1]);
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%
```

NE_insert_priority_queue

```
function queue_out =
NE_insert_priority_queue(T,queue)
```

```
% NE_insert_priority_queue- insert triangle into
priority queue
%     priority is distance of midpoint from PD
origin
% On input:
%     T (3x2 array): triangle vertexes
%     queue (queue struct vector): triangle queue
%       (k).corners (3x2 array): vertexes of
triangle
% On output:
%     queue_out (queue struct vector): triangle
queue with T inserted
%       (k).corners (3x2 array): vertexes of
triangle
% Call:
%     q = NE_insert_priority_queue(T,q);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_int_E2_2circles

```
function [ip1,ip2,status] =
NE_int_E2_2circles(circle1,circle2)
% NE_int_E2_2circles - find R^2 intersection points
of 2 circles
% On input:
%     circle1 (1x3 vector): center and radius of
circle 1
%     circle2 (1x3 vector): center and radius of
circle 2
% On output:
%     ip1 (1x2 vector): intersection point 1
%     ip2 (1x2 vector): intersection point 2
%     status (int): 0 if no intersection
```

```
%                    1 if 1 point
%                    2 if 2 points;
%                    3 if same circle
% Call:
%     [p1,p2,st] =
NE_int_E2_2circles([0,0,2],[1,0,1]);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_int_2circles

```
function [ip1,ip2,status] =
NE_int_2circles(circle1,circle2)
% NE_int_2circles - find intersection points of 2
circles
% On input:
%     circle1 (1x3 vector): center and radius of
circle 1
%     circle2 (1x3 vector): center and radius of
circle 2
% On output:
%     ip1 (1x2 vector): intersection point 1
%     ip2 (1x2 vector): intersection point 2
%     status (int): 0 if no intersection
%                    1 if 1 point
%                    2 if 2 points;
%                    3 if same circle
% Call:
%     [p1,p2,st] =
NE_int_2circles([0,0,2],[1,0,1]);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
```

%

NE_int_line_circle

```
function [ip1,ip2,status] =
NE_int_line_circle(P,Q,circle)
% NE_int_line_circle - find intersection points of
line and circle
% On input:
%     P (1x2 vector): point 1 defining line
%     Q (1x2 vector): point 2 defining line
%     circle (1x5 vector): center, radius and
circle pt
% On output:
%     ip1 (1x2 vector): intersection point 1
%     ip2 (1x2 vector): intersection point 2
%     status (int): 0 if no intersection; 1 if
single point; 2 if 2 points
% Call:
%     [p1,p2,s] = NE_int_line_circle([-
0.5,0.6],[0.5,0.6],[0,0,1,1,0]);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_inversion_all

```
function ptsi = NE_inversion_all(P,Q,circle,pts)
% NE_inversion_all - invert a set of points through
a circle
%                  2 of the pts should be on the
circle
% On input:
%     circle (1x3 vector): center, radius
```

```
%     pts (nx2 array): x,y locations in the
Poincare disk
% On output:
%     ptsi (nx2 array): inversion points
% Call:
%     ptsi = NE_inversion_all(cir1,pts);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_inversion_experiment1

```
function inverted =
NE_inversion_experiment1(max_count)
% NE_inversion_experiment1 - tile PD with regular
pentagon
%    put pentagon in center of PD and invert
through sides recursively
% On input:
%     max_count (int): maximum number of pentagons
to create (non-uniquely)
% On output:
%     inverted (nx10 array): linearized set of
unique pentagons created
% Call:
%     inv = NE_inversion_experiment1(40);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_inversion_experiment2

```
function inverted =
NE_inversion_experiment2(max_count)
% NE_inversion_experiment2 - tile PD with 45-degree
triangles
%     put triangles in center of PD and invert
through sides recursively
% On input:
%     max_count (int): number of unique triangles
to generate
% On output:
%     inverted (nx6 array): linearized list of
existing triangles
% Call:
%     invT = NE_inversion_experiment2(16);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_norm_PD

```
function d = NE_norm_PD(pt)
% NE_norm_PD - length of vector (from origin) in
Poicare disk
%     pt1 (1x2 vector): x,y values of point
% On output:
%     d (float): distance from origin to point
% Call:
%     d = NE_norm_PD([0;0.9]);
% Author:
%     T. Henderson
```

```
%      UU
%      Summer 2023
%
```

NE_orthog_circle

```
function circle = NE_orthog_circle(P,Q)
% NE_orthog_circle - find orthogonal circle to unit
circle through P & Q
% On input:
%      P (1x2 vector): point inside unit disk
%      Q (1x2 vector): point inside unit disck
distinct from P
% On output:
%      circle (1x3 vector): center and radius of
orthogonal circle
% Call:
%      circle =
NE_orthog_circle([0.2,0.8],[0.4,0.6]);
% Author:
%      T. Henderson
%      UU
%      Summer 2023
%
```

NE_PD_seg_pts

```
function pts = NE_PD_seg_pts(pt1,pt2)
% NE_PD_seg_pts - return set of points along PD
segment from pt1 to pt2
% On input:
%      pt1 (1x2 vector): first point
%      pt2 (1x2 vector): second point
% On output:
%      pts (nx2 array): points along segment
```

```
% Call:
%     pt = NE_PD_seg_pts([-0.5,.5],[0.5,0.5]);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_pts2PDline

```
function [C,r,z1,z4] = NE_ptsPD2line(P,Q)
% NE_ptsPD2line - find geodesic through 2 Poincare
disk points
% On input:
%     P (1x2 vector): point 1
%     Q (1x2 vector): point2
% On output:
%     C (1x2 vector) center of geodesic circle or
non-origin point
%                     if points lie on diagonal
%     r (float): radius (Inf if diagonal geodesic)
%     z1 (1x2 vector): point on unit disk boundary
closer to P
%     z4 (1x2 vector): point on unit disk boundary
closer to Q
%       i.e., order is: z1 P, Q, z4  for cross
ratio calculation
% Call:
%     [C,r,z1,z4] = NE_ptsPD2line(P,Q);
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_test_angle_2circles

```
function NE_test_angle_2circles
% NE_test_angle_2circles - test all possible PD
%                          circle angle combos
% On input:
%     N/A
% On output:
%     figure shows results
% Call:
%     NE_test_angle_2circles
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_test_int_E2_2circles

```
function NE_test_int_E2_2circles
% NE_test_int_E2_2circles - test all possible
circle interactions in R^2
% On input:
%     N/A
% On output:
%     figure with circles and results
% Call:
%     NE_test_int_E2_2circls;
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```

NE_test_PD_circle

```
function A = NE_test_PD_circle
% NE_test_PD_circle - look at circles interior to
PD
%   the definition of a circle in PD is the set of
points that
%   are at a constant distance from a given center.
% On input:
%     N/A
% On output:
%     N/A   - just put break points and examine
data
% Call:
%     NE_test_PD_circle
% Author:
%     T. Henderson
%     UU
%     Summer 2023
%
```