Appendix C

Algorithm for Peptide Clustering

C.1 INTRODUCTION

The one-bead-one-compound *in situ* click screening approach often yields a number of peptide "hits." Rather than synthesizing and vetting each individual sequence, it can be useful to group the results into clusters of peptides based on physical properties, and then to choose a representative peptide from each cluster to scale up and test. This appendix provides an example clustering algorithm that uses the concept of persistence homology to group peptides by hydrophobicity, isoelectric point, and residue weight. The grouping is amino acid order dependent.

C.2 PERSISTENCE CLUSTERING

Persistence states that structures that persist at multiple scales are most likely to be real. For example, if an image is sampled by using a small number of points, persistence can be used to determine properties of the original shape. The points are grouped by drawing a disc of radius ε around each point. Each collection of overlapping discs is considered a cluster. These clusters provide one possible interpretation of the original data.

To obtain a more complete analysis of the data, ε is continuously varied and the clusters tracked as the discs grow around the points. As discs from one cluster begin to overlap with discs from another cluster, the two initial clusters "die" and a new merged cluster is "born." The lifetime of a cluster is the time between a cluster's birth and death. Thus, groups of points that are far from other points will have a long lifetime, which indicates that they are more meaningful clusters.

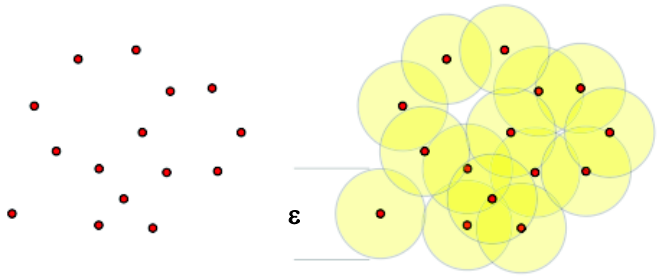

Figure C.1. Discs formed around points with growing radius ε. As the discs around points overlap, new clusters form that include points rooting the newly touching discs. Eventually all the discs will touch, and the cluster containing all points will continue for infinite "time." Adapted from [1].

C.3 ALGORITHM DESCRIPTION

The MatLab function BasicParam takes as input text files that contain a list of sequences and the number *n* of amino acids per sequence. It then uses hydrophobicity, isoelectric point, and residue weight to construct a point in 3*n* dimensional for each sequence. To prevent any of the physical parameters from overly contributing to the clustering based on their absolute amounts, they are all scaled by the inverse of their average. Once each sequence is represented by a point, the Euclidian pair wise distances between the points are computed. Initially, each point is its own cluster. Then, the two points with the smallest pair wise distance are merged. The time of death of the initial point clusters, and the time of birth of the new merged cluster, is set to the distance between those two points (see Figure B.2). Clusters containing points with the next least distance continue to iteratively merge until all points in the data set exist as a single cluster. Additionally, a covariance matrix was constructed and diagonalized to find the eigenvectors and eigenvalues that characterize the sequence space.

C.4 OUTPUT

The function outputs a list of clusters and their associated lifetimes, along with a plot of the input sequences projected onto the top two eigenvectors. The complete list of eigenvectors is also plotted against the associated eigenvalues, and the top four eigenvectors are decomposed into their individual components as charts. The output from clustering the hits obtained in the A22/4B3 screen described in Chapter 2 is shown in Table B.1 and Figures B.3 – B.5.

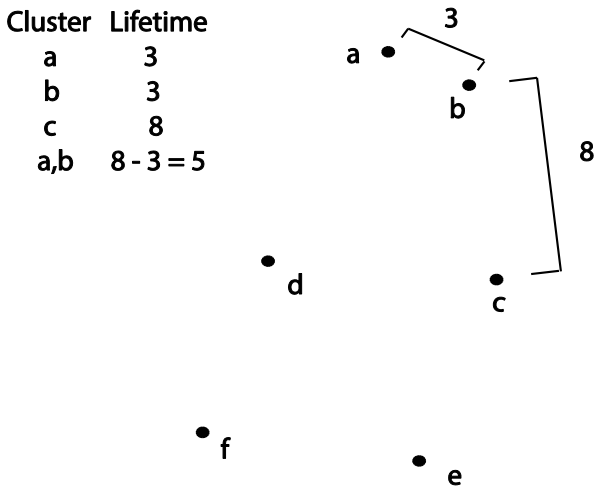

Figure C.2. Illustrative example of how lifetimes of clusters are computed. The lifetimes of clusters “a” and “b” are 3, because they are closer to each other than any other point and that is the distance between them. 3 then becomes the birth time for cluster “ab,” and the lifetime of “ab” is 5 because it dies at time 8, when the cluster “abc” is born.

3.153	ffftk														
2.908	qpidq														
2.418	vgeit														
2.331	qspwl														
2.331	qsdwl														
2.281	ltsry														
2.281	aapsl														
2.249	kysfq														
2.195	qndwk														
2.195	eqtfd														
1.928	kqdtq														
1.928	tgnek														
1.815	kidrv														
1.728	kwtkf														
1.246	diksp														
1.246	eihny														
0.702	diksp	eihny	kidrv	kwtkf											
0.482	diksp	eihny													
0.375	qspwl	qsdwl													
0.317	kqdtq	tgnek													
0.293	ltsry	diksp	eihny	kidrv	aapsl	kwtkf									
0.288	qndwk	eqtfd	kqdtq	tgnek	kysfq	vgeit									
0.245	qndwk	ltsry	diksp	eihny	qpidq	kidrv	qspwl	qsdwl	eqtfd	aapsl	kqdtq	kwtkf	tgnek	kysfq	vgeit
0.237	ltsry	aapsl													
0.169	qndwk	eqtfd	kqdtq	tgnek	kysfq										
0.105	qndwk	qspwl	qsdwl	eqtfd	kqdtq	tgnek	kysfq	vgeit							
0.097	qndwk	ltsry	diksp	eihny	kidrv	qspwl	qsdwl	eqtfd	aapsl	kqdtq	kwtkf	tgnek	kysfq	vgeit	
0.087	diksp	eihny	kwtkf												
0.050	qndwk	eqtfd													
0.004	qndwk	eqtfd	kqdtq	tgnek											

Table C.1. List of clusters and their associated lifetimes calculated for the hits resulting from the A22/4B3 screen. The peptides input to the algorithm were qndwk, ltsry, diksp, eihny, qpidq, kidrv, qspwl, qsdwl, eqtfd, aapsl, kqdtq, kwtkf, tgnek, kysfq, ffftk, and vgeit.

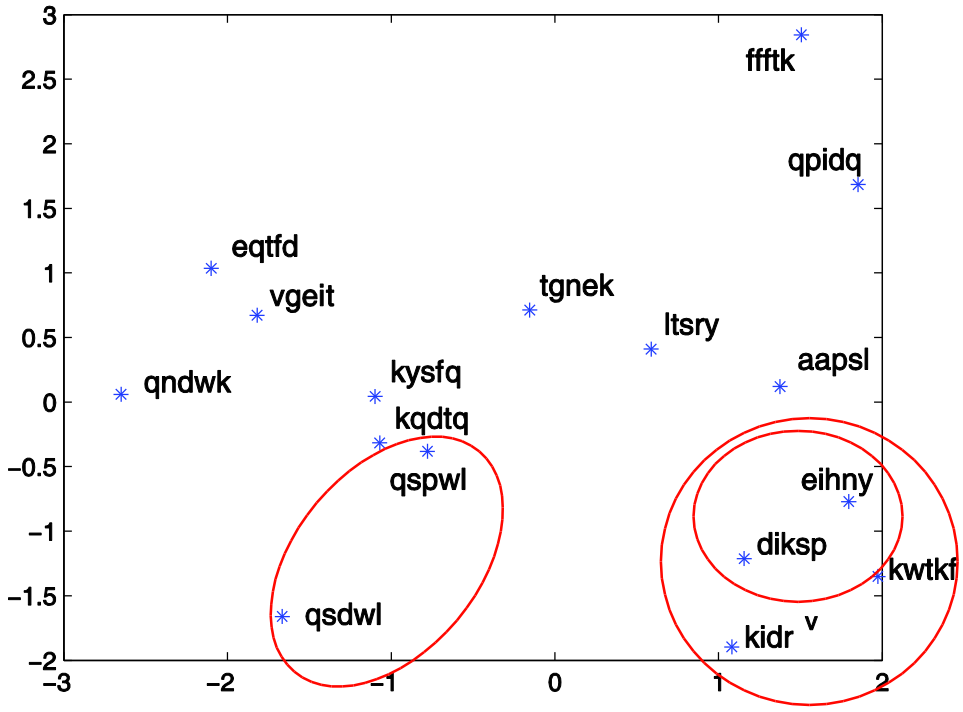

Figure C.3. Projection of the peptide "points" onto the top two eigenvectors taken from the diagonalization of the covariance matrix. The clusters containing more than one point with the three longest lifetimes from Table B.1 are circled.

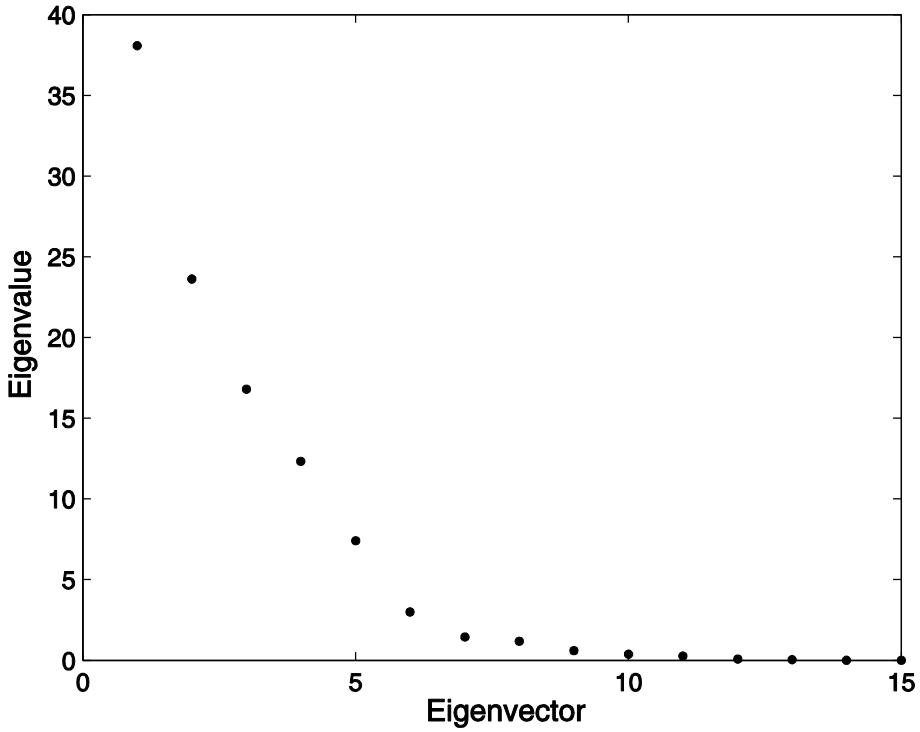

Figure C.4. Plot of eigenvectors vs. their associated eigenvalues. This allows users to know how many relevant eigenvectors describe the set of input peptides.

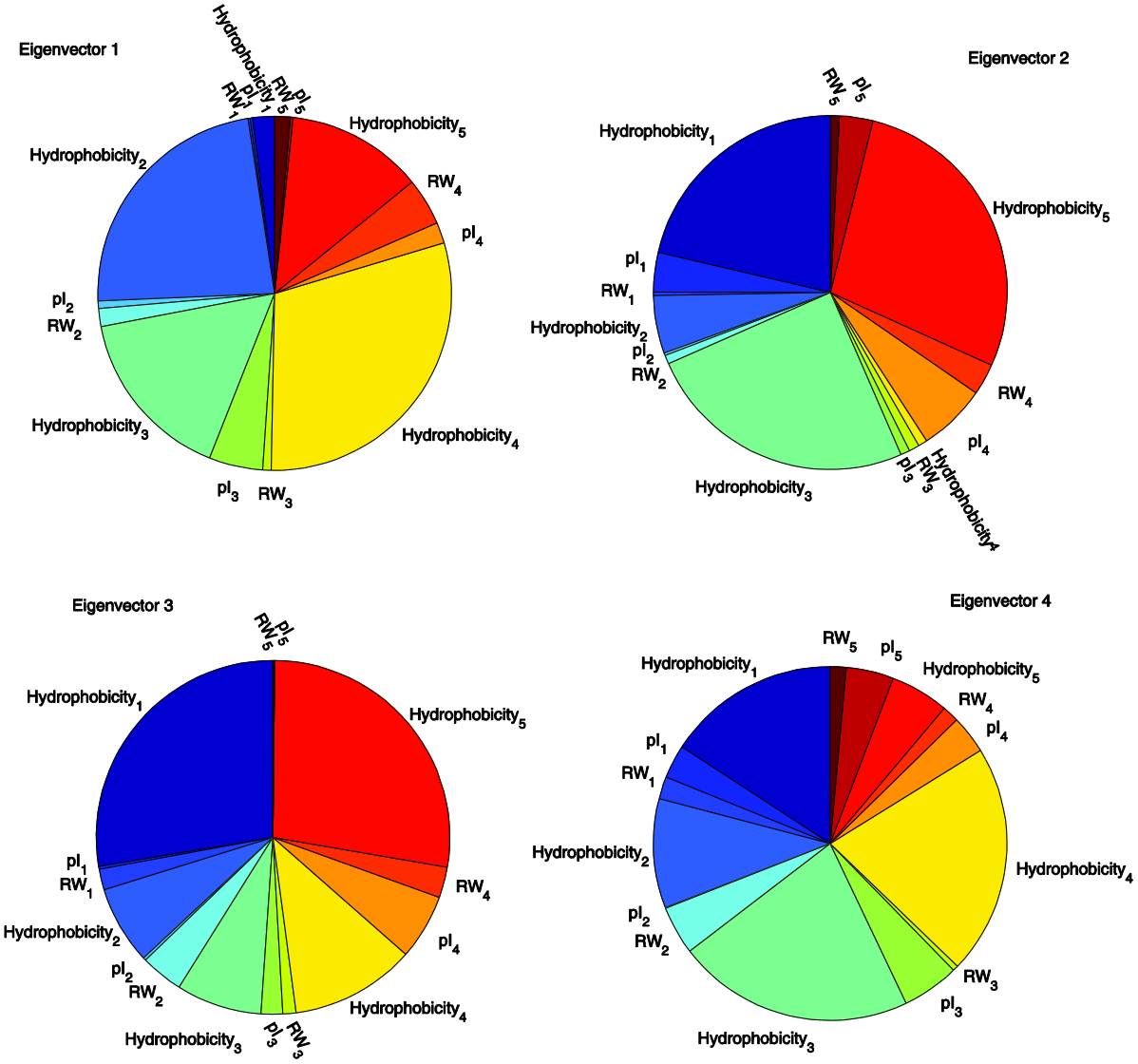

Figure C.5. Visual representation of the components of the top four eigenvectors. This shows what physical characteristics are dominant when describing the list of input peptides. The order of the amino acids is preserved during clustering, so Hydrophobicity_1 refers to the hydrophobicity of the N-terminal amino acid of the peptides. For this set, the top four eigenvectors have large hydrophobicity components, though not all from the same amino acid index.

C.5 FUNCTION CODE

```
function BasicParam(filename, pepLength)

%Feb 8, 2013
%Physical parameters come from the wwHydrophobicity scale, Sigma PI
values,
%and resudue weight.
%Note: hydrophobicities are at pH 7 except for proline, which was
estimated
%from an InDi (or AN SD) measurement
%Note: the number of traits is hard coded in, so that has to be changed
%every time you mess with the parameters.

%http://www.sigmaaldrich.com/life-science/metabolomics/learning-
center/amin
%o-acid-reference-chart.html

Hydr =
1/((41+49+55+31+100+0+8+99+23+97+74+28+50+10+14+5+13+76+97+63)/20);
PI =
1/((6+5.07+2.77+3.22+5.48+5.97+7.59+6.02+9.74+5.98+5.74+5.41+6.3+5.56+1
0.76+5.68+5.6+5.96+5.89+5.66)/20);
RW =
1/((71.08+103.15+115.09+129.12+147.18+57.05+137.14+113.16+128.18+113.16
+131.2+114.11+97.12+128.13+156.19+87.07+101.11+99.13+186.22+163.18)/20)
;
scaling = [Hydr PI RW];
%scaling = 1;

%[Hydrophobicity pI ResidueWeight]

A = [41 6.00 71.08].*scaling;
C = [49 5.07 103.15].*scaling;
D = [-55 2.77 115.09].*scaling;
E = [-31 3.22 129.12].*scaling;
F = [100 5.48 147.18].*scaling;
G = [0 5.97 57.05].*scaling;
H = [8 7.59 137.14].*scaling;
I = [99 6.02 113.16].*scaling;
K = [-23 9.74 128.18].*scaling;
L = [97 5.98 113.16].*scaling;
M = [74 5.74 131.20].*scaling;
N = [-28 5.41 114.11].*scaling;
P = [50 6.30 97.12].*scaling;
Q = [-10 5.56 128.13].*scaling;
R = [-14 10.76 156.19].*scaling;
S = [-5 5.68 87.07].*scaling;
T = [13 5.60 101.11].*scaling;
V = [76 5.96 99.13].*scaling;
W = [97 5.89 186.22].*scaling;
Y = [63 5.66 163.18].*scaling;

%Read in sequences from text file function argument, and determine the
```

```
%numberof sequences
sequences = textread(filename,'%s', -1);
nseq = length(sequences);

%Initialize matrix to hold each sequence vector
ntrait = 3;
Z=zeros(nseq,pepLength*ntrait);

%For each individual sequence assign a vector based on Grantham Values
for j = 1:nseq
    seq = sequences{j};
    o = [];

    for i=1:length(seq)

    switch seq(i)
        case 's'
            o = [o S];
        case 'r'
            o = [o R];
        case 'l'
            o = [o L];
        case 'p'
            o = [o P];
        case 't'
            o = [o T];
        case 'a'
            o = [o A];
        case 'v'
            o = [o V];
        case 'g'
            o = [o G];
        case 'i'
            o = [o I];
        case 'f'
            o = [o F];
        case 'y'
            o = [o Y];
        case 'c'
            o = [o C];
        case 'h'
            o = [o H];
        case 'q'
            o = [o Q];
        case 'n'
            o = [o N];
        case 'k'
            o = [o K];
        case 'd'
            o = [o D];
        case 'e'
            o = [o E];
        case 'm'
            o = [o M];
        case 'w'
            o = [o W];
```

```
    end

    end
  Z(j,:) = o;
end

%Compute pairwise distances between each "sequence point" given by
%a row in the Z matrix and put in array n.  Also, create a 2x#pairs
matrix
%cataloging pair indices to go along with distances.
n=[];
Idx=[];
k=1;
for i = 1:nseq
    for j = i+1:nseq
        n(k) = norm(Z(i,:)-Z(j,:));
        Idx(1,k) = i;
        Idx(2,k) = j;
        k=k+1;
    end
end

%Sort the array of distances from smallest to largest and use returned
%permutation to analogously sort pair indices.
[N,O] = sort(n);
Idx = Idx(:,O);

%Create array of parents of "sequence points"
p = [1:nseq];

%Create an array of cluster birth
tb = zeros(1,nseq);

% %Union the points with distance smaller than 200
% for k = 1:length(N)
%     if(N(k)>= 200)
%         %sprintf('stopping at distance %f\n', N(k))
%         break;
%     end
%   %sprintf('Unioning %f and %f with distance
%f\n',Idx(1,k),Idx(2,k),N(k))
%   p = Union(Idx(1,k),Idx(2,k),p);
% end

% %'Find' all points to see which ones share a root after Union and put
into
% %array 'root'
% for i = 1:nseq
%     j = Find(i,p);
%     root(i) = j;
% end

% [R,O_r] = sort(root);
% sequences(O_r);
```

```
%Itteratively 'Union' clusters containing the points with the next
%least distance

sets = {};
ages = [];
birth = [];
death = [];
for i = 1:length(N)

    %Find roots of the two sets being joined
    r1 = Find(Idx(1,i),p);
    r2 = Find(Idx(2,i),p);

    %Set the current time to the distance between the two merging
clusters
    t = N(i);

    %Check if the two sequences are already in the same cluster
    if r1 ~= r2

    %Save the sequences in the two merging clusters
    %disp('====================================');
    set1 = MkSet(r1,nseq,p,sequences);
    set2 = MkSet(r2,nseq,p,sequences);

    sets(end+1) = {set1};
    sets(end+1) = {set2};

    %Compute and save the ages of the merging clusters
    age1 = t - tb(r1);
    age2 = t - tb(r2);

    ages = [ages age1];
    ages = [ages age2];

    %Keep running tally of birth/death times for persistence barcode
    birth = [birth tb(r1)];
    birth = [birth tb(r2)];

    death = [birth t];
    death = [birth t];

    %Union the two clusters
    p = Union(Idx(1,i),Idx(2,i),p);

    %Save birth time for new cluster
    tb(r2) = t;

    end
end
birth;
death;
%Sort ages from largest to smallest, accordingly rearrainge sets of
%sequence names
```

```
[Ages,O_A] = sort(ages, 'descend');
Sets = sets(O_A);

%Read out vectors of ages and sets
 %disp('-------------------------------------------------------')
 %DispSet(sets, ages);
% disp('-------------------------------------------------------')
 DispSet(Sets, Ages);
 disp('-------------------------------------------------------')

%Plot "persistence barcode" to see if any clustering is actually
present



%Do PCA analysis to see if there exist any major axes along which the
%sequence data falls.

m = size(Z);

%Center data
center = zeros(1,m(2));
for i = 1:nseq
center = center + Z(i,:);
end
center = center/nseq;
Zc = zeros(m);
for i = 1:nseq
Zc(i,:) = Z(i,:) - center;
end

%Create covarience matrix
covar = zeros(m(2), m(2));
CoVar = zeros(m(2), m(2));
for i = 1:nseq
    covar = Zc(i,:)'*Zc(i,:);
    CoVar = CoVar + covar;
end

%Diagonalize covarience matrix to find largest eigenvalues and
%corresponding eigenvectors
[vec,val] = eig(CoVar);
CoVar*vec - vec*val;

%Plot eigenvalues and display eigenvectors associated with two largest
%values.  Also label eigenvectors with next two largest eigenvalues to
%decompose into pie charts to examine components.
Diagval = diag(val);
[Val,O_val] = sort(Diagval, 'descend');
Vec = vec(:,O_val);
plot(Val, '.k', 'MarkerSize',10);
eig1 = Vec(:,1); eig2 = Vec(:,2); eig3 = Vec(:,3); eig4 = Vec(:,4);

%Project sequence points onto the two largest eigenvectors and plot
Projection(Vec(:,1), Vec(:,2), Zc, sequences, nseq);
```

```
%Create pie chart to examine the components of the four eigenvectors
with
%the largest eigenvalues
labels = {'Hydrophobicity_1', 'pI_1', 'RW_1','Hydrophobicity_2',
'pI_2',...
    'RW_2', 'Hydrophobicity_3', 'pI_3', 'RW_3', 'Hydrophobicity_4',
'pI_4',...
    'RW_4','Hydrophobicity_5', 'pI_5', 'RW_5'};
figure
pie(abs(eig1), labels)
title('Eigenvector 1')

figure
pie(abs(eig2), labels)
title('Eigenvector 2')

figure
pie(abs(eig3), labels)
title('Eigenvector 3')

figure
pie(abs(eig4), labels)
title('Eigenvector 4')

end


%-------------------------------------------------------------------------
--%

function y = Find(x,p)
    y = x;
    while(p(y)~=y)
        y = p(y);
    end
end

function p = Union(x1,x2,p)
    r1 = Find(x1,p);
    r2 = Find(x2,p);
    p(r1) = r2;
end

function Set = MkSet(r1,nseq,p,seq)
    Set = {};
    for i = 1:nseq
        r = Find(i,p);
        if r == r1
            Set(end+1) = seq(i);
        end
    end
end

function DispSet(Sets, Ages)
```

```
    for i = 1:length(Sets)
        Si = Sets{i};
        str = sprintf('%f\t',Ages(i));
        for j=1:length(Si)
            str = sprintf('%s %s\t', str, char(Si{j}));
        end
        disp(str);
    end
end

function Projection(eig1, eig2, Zc, sequences, nseq)
x = [];
y = [];
    for i = 1:nseq
        x = [x Zc(i,:)*eig1];
        y = [y Zc(i,:)*eig2];
    end
    figure;
    plot(x, y, '*')
    dx = .2; dy = 0.2;
    text(x+dx, y+dy, sequences);
end
```

C.6 CONCLUSION

This appendix describes a clustering algorithm designed to help characterize groups of peptide hits from *in situ* click screens. The objective is to provide guidance about which peptides are useful to scale up and assay, and give information about the physical characteristics that predominately describe the differences between input peptides.

C.7 ACKNOWLEGDEMENTS

I would like to thank Keenan Crane for introducing me to the concept of persistent homology, and for his incredibly helpful guidance while developing this algorithm.

C.8 REFERENCES

1. Ghrist R (2008) Barcodes: the persistent topology of data. Bulletin of the American Mathematical Society 45: 61-75.