1 Scoring matrix comparison

Goal: use matplotl i b to plot heatmaps for compar i ng BLOSUM matr i ces

1.1 Load matrices from NCBI BLAST

What do they look l i ke?

```
!cat blosum45.txt
```

```
#  Matrix made by matblas from blosum45.iij
#  * column uses minimum score
#  BLOSUM Clustered Scoring Matrix in 1/3 Bit Units
#  Blocks Database = /data/blocks_5.0/blocks.dat
#  Cluster Percentage: >= 45
#  Entropy =   0.3795, Expected =  -0.2789
l  A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  5 -2 -1 -2 -1 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -2 -2  0 -1 -1 -1 -5
R -2  7  0 -1 -3  1  0 -2  0 -3 -2  3 -1 -2 -2 -1 -1 -2 -1 -2 -1  0 -1 -5
N -1  0  6  2 -2  0  0  0  1 -2 -3  0 -2 -2 -2  1  0 -4 -2 -3  4  0 -1 -5
D -2 -1  2  7 -3  0  2 -1  0 -4 -3  0 -3 -4 -1  0 -1 -4 -2 -3  5  1 -1 -5
C -1 -3 -2 -3 12 -3 -3 -3 -3 -3 -2 -3 -2 -2 -4 -1 -1 -5 -3 -1 -2 -3 -1 -5
Q -1  1  0  0 -3  6  2 -2  1 -2 -2  1  0 -4 -1  0 -1 -2 -1 -3  0  4 -1 -5
E -1  0  0  2 -3  2  6 -2  0 -3 -2  1 -2 -3  0  0 -1 -3 -2 -3  1  4 -1 -5
G  0 -2  0 -1 -3 -2 -2  7 -2 -4 -3 -2 -2 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -5
H -2  0  1  0 -3  1  0 -2 10 -3 -2 -1  0 -2 -2 -1 -2 -3  2 -3  0  0 -1 -5
I -1 -3 -2 -4 -3 -2 -3 -4 -3  5  2 -3  2  0 -2 -2 -1 -2  0  3 -3 -3 -1 -5
L -1 -2 -3 -3 -2 -2 -2 -3 -2  2  5 -3  2  1 -3 -3 -1 -2  0  1 -3 -2 -1 -5
K -1  3  0  0 -3  1  1 -2 -1 -3 -3  5 -1 -3 -1 -1 -1 -2 -1 -2  0  1 -1 -5
M -1 -1 -2 -3 -2  0 -2 -2  0  2  2 -1  6  0 -2 -2 -1 -2  0  1 -2 -1 -1 -5
F -2 -2 -2 -4 -2 -4 -3 -3 -2  0  1 -3  0  8 -3 -2 -1  1  3  0 -3 -3 -1 -5
P -1 -2 -2 -1 -4 -1  0 -2 -2 -2 -3 -1 -2 -3  9 -1 -1 -3 -3 -3 -2 -1 -1 -5
S  1 -1  1  0 -1  0  0  0 -1 -2 -3 -1 -2 -2 -1  4  2 -4 -2 -1  0  0 -1 -5
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -1 -1  2  5 -3 -1  0  0 -1 -1 -5
W -2 -2 -4 -4 -5 -2 -3 -2 -3 -2 -2 -2 -2  1 -3 -4 -3 15  3 -3 -4 -2 -1 -5
Y -2 -1 -2 -2 -3 -1 -2 -3  2  0  0 -1  0  3 -3 -2 -1  3  8 -1 -2 -2 -1 -5
V  0 -2 -3 -3 -1 -3 -3 -3 -3  3  1 -2  1  0 -3 -1  0 -3 -1  5 -3 -3 -1 -5
B -1 -1  4  5 -2  0  1 -1  0 -3 -3  0 -2 -3 -2  0  0 -4 -2 -3  4  2 -1 -5
Z -1  0  0  1 -3  4  4 -2  0 -3 -2  1 -1 -3 -1  0 -1 -2 -2 -3  2  4 -1 -5
X -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -5
* -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5  1
```

```
def load_blastmatrix(fname):
    """Parse a scoring matrix in NCBI format and return
    it as a two-dimensional dictionary"""
    fp = open(fname)
    # kill file header
    for line in fp:
        if(not line.startswith("#")):
            break
    # index column headers
```

```
    i2a = dict((i,a.strip()) for (i,a) in enumerate(line.split()))
    # parse scores
    b = {}
    for line in fp:
        w = line.rstrip("\r\n").split()
        # row header
        a = w[0]
        # row values
        d = {}
        for (i, j) in enumerate(w):
            if(i == 0):
                continue
            d[i2a[i]] = int(j)
        b[a] = d
    return b
```

```
blosum = [load_blastmatrix("blosum%s.txt" % i) for i in (45,62,80)]
```

1.2 Convert to numpy arrays

```
alpha_aa = "ACDEFGHIKLMNPQRSTVWY"
```

```
scores = [array([[b[i][j] for j in alpha_aa] for i in alpha_aa]) for b in blosum]
```

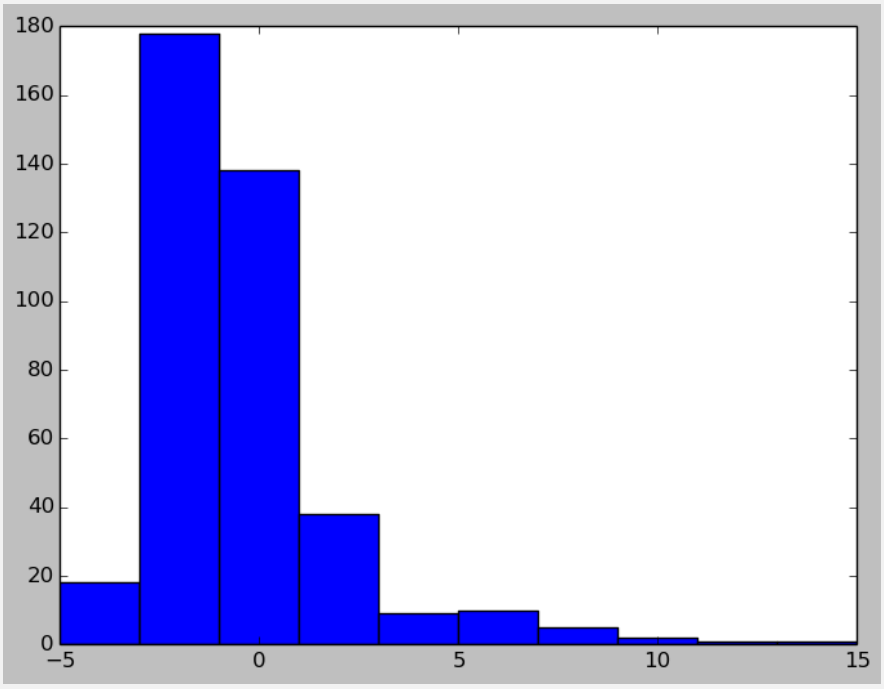
```
scores[0]
```

```
array([[ 5, -1, -2, -1, -2,  0, -2, -1, -1, -1, -1, -1, -1, -2,  1,  0,
         0, -2, -2],
       [-1, 12, -3, -3, -2, -3, -3, -3, -3, -2, -2, -2, -4, -3, -3, -1, -1,
        -1, -5, -3],
       [-2, -3,  7,  2, -4, -1,  0, -4,  0, -3, -3,  2, -1,  0, -1,  0, -1,
        -3, -4, -2],
       [-1, -3,  2,  6, -3, -2,  0, -3,  1, -2, -2,  0,  0,  2,  0,  0, -1,
        -3, -3, -2],
       [-2, -2, -4, -3,  8, -3, -2,  0, -3,  1,  0, -2, -3, -4, -2, -2, -1,
         0,  1,  3],
       [ 0, -3, -1, -2, -3,  7, -2, -4, -2, -3, -2,  0, -2, -2, -2,  0, -2,
        -3, -2, -3],
       [-2, -3,  0,  0, -2, -2, 10, -3, -1, -2,  0,  1, -2,  1,  0, -1, -2,
        -3, -3,  2],
       [-1, -3, -4, -3,  0, -4, -3,  5, -3,  2,  2, -2, -2, -2, -3, -2, -1,
         3, -2,  0],
       [-1, -3,  0,  1, -3, -2, -1, -3,  5, -3, -1,  0, -1,  1,  3, -1, -1,
        -2, -2, -1],
       [-1, -2, -3, -2,  1, -3, -2,  2, -3,  5,  2, -3, -3, -2, -2, -3, -1,
         1, -2,  0],
       [-1, -2, -3, -2,  0, -2,  0,  2, -1,  2,  6, -2, -2,  0, -1, -2, -1,
         1, -2,  0],
       [-1, -2,  2,  0, -2,  0,  1, -2,  0, -3, -2,  6, -2,  0,  0,  1,  0,
```

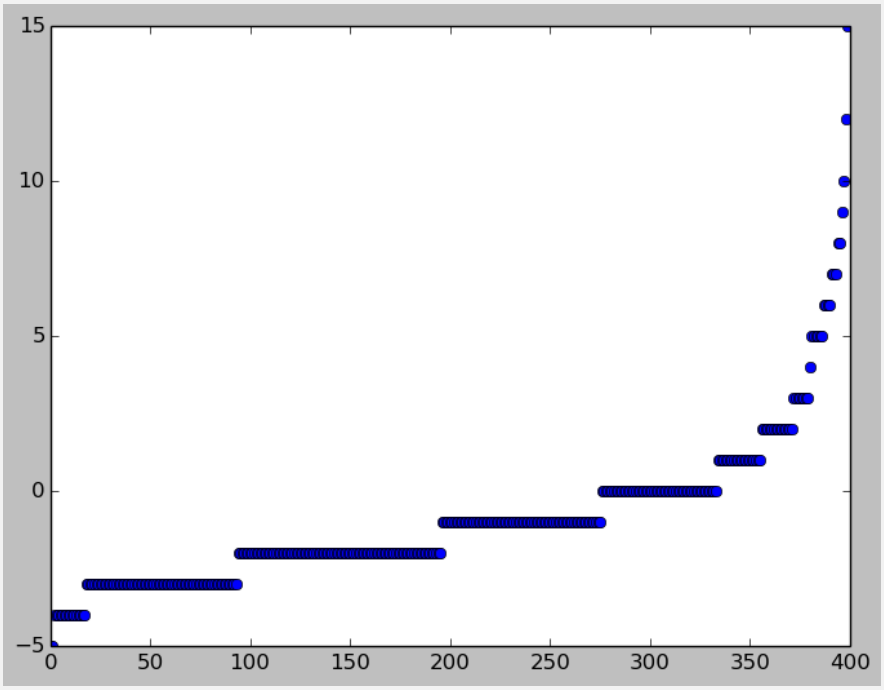
```
 -3, -4, -2],
[-1, -4, -1,  0, -3, -2, -2, -2, -1, -3, -2, -2,  9, -1, -2, -1, -1,
 -3, -3, -3],
[-1, -3,  0,  2, -4, -2,  1, -2,  1, -2,  0,  0, -1,  6,  1,  0, -1,
 -3, -2, -1],
[-2, -3, -1,  0, -2, -2,  0, -3,  3, -2, -1,  0, -2,  1,  7, -1, -1,
 -2, -2, -1],
[ 1, -1,  0,  0, -2,  0, -1, -2, -1, -3, -2,  1, -1,  0, -1,  4,  2,
 -1, -4, -2],
[ 0, -1, -1, -1, -1, -2, -2, -1, -1, -1, -1,  0, -1, -1, -1,  2,  5,
  0, -3, -1],
[ 0, -1, -3, -3,  0, -3, -3,  3, -2,  1,  1, -3, -3, -3, -2, -1,  0,
  5, -3, -1],
[-2, -5, -4, -3,  1, -2, -3, -2, -2, -2, -2, -4, -3, -2, -2, -4, -3,
 -3, 15,  3],
[-2, -3, -2, -2,  3, -3,  2,  0, -1,  0,  0, -2, -3, -1, -1, -2, -1,
 -1,  3,  8]])
```

1.3 Choose an appropriate scale

```
a = scores[0].reshape((400,))
fig = figure()
h = hist(a)
display(fig)
```



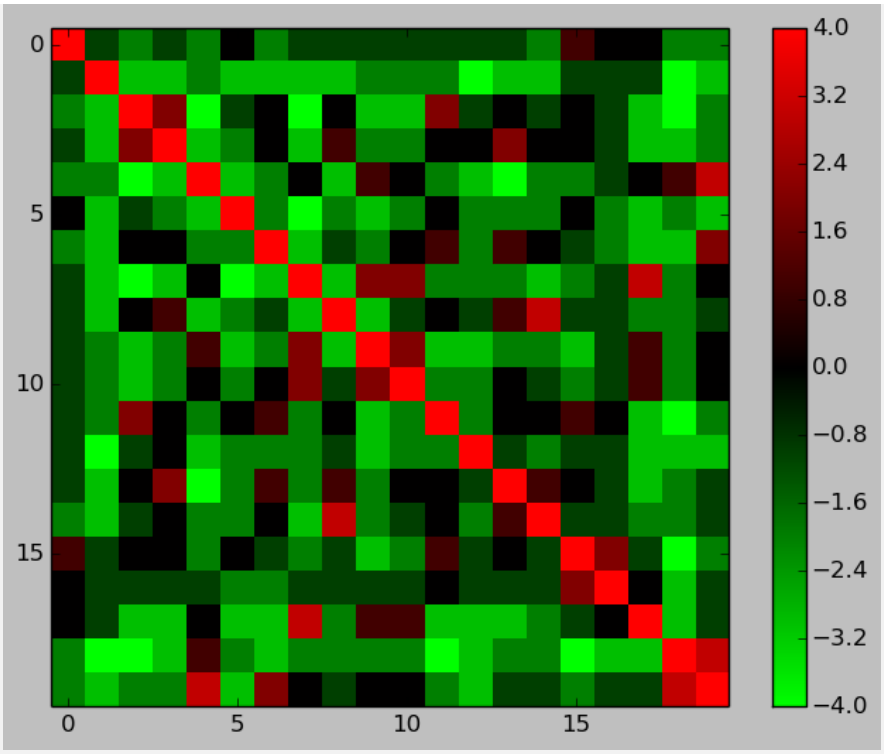

```
fig = figure()
plot(sorted(a),"bo")
display(fig)
```

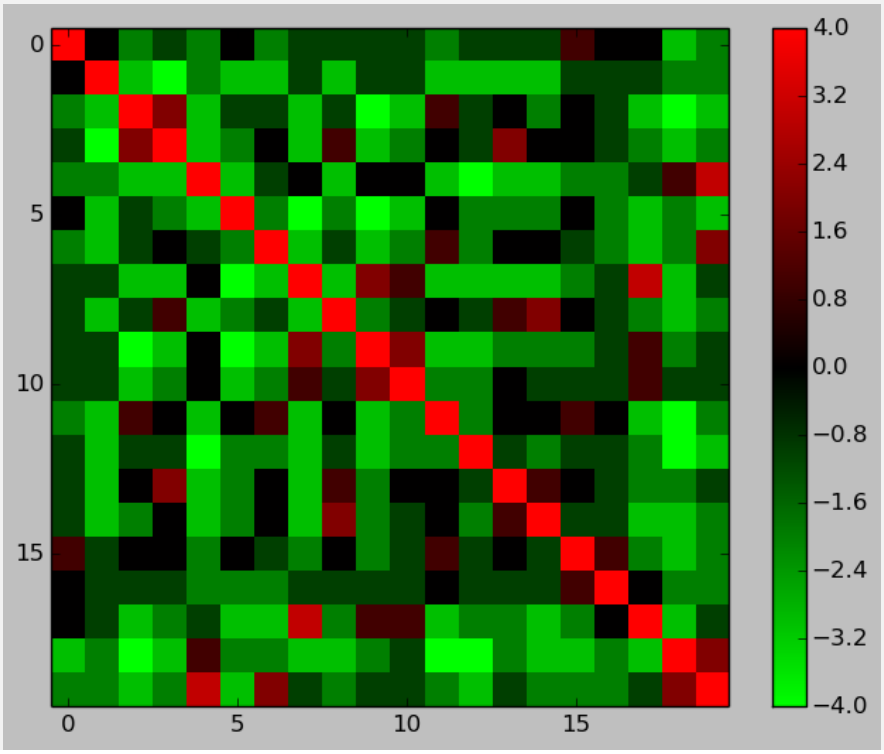


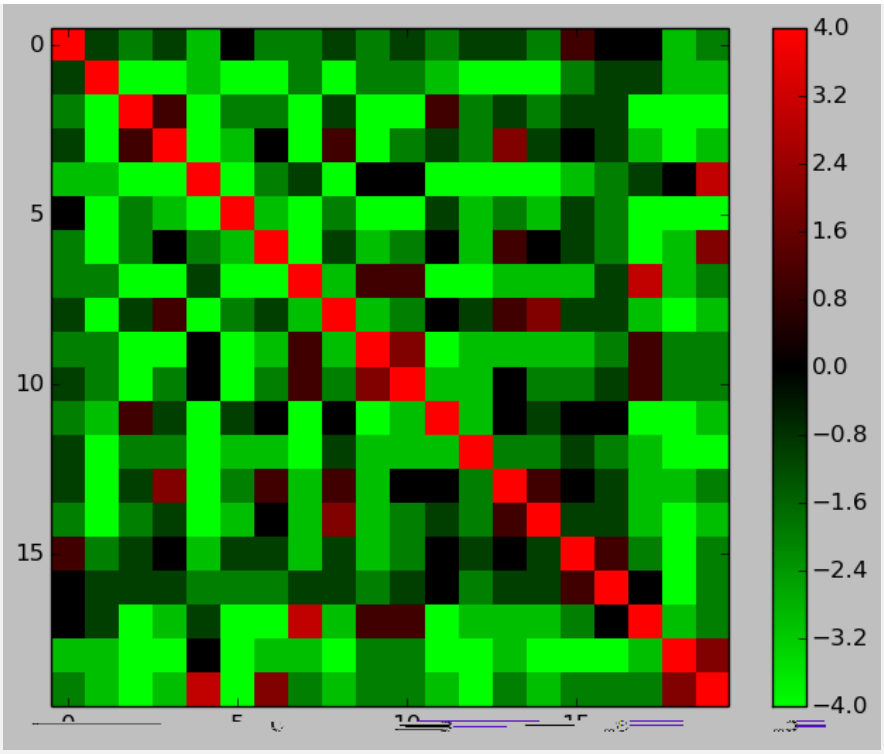
1.4 Generate comparative plots

```
# Simple green->red non-overlapping gradient
cdict = {"red":((0.,0.,0.),(.5,0.,0.),(1.,1.,1.)),
         "green":((0.,1.,1.),(.5,0.,0.),(1.,0.,0.)),
         "blue":((0.,0.,0.),(.5,0.,0.),(1.,0.,0.))}
# map gradient to 256 actual RGBA values
myc = matplotlib.colors.LinearSegmentedColormap("myc",cdict,256)
for i in scores:
    fig = figure()
    # expand color gradient from (0,1) to (-4,4)
    imshow(i, cmap = myc, clim = (-4,4), interpolation = "nearest")
    colorbar()
```

```
display(*(getfigs()[-3:]))
```


0
5
10
15
0
5
10
15
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0


0
5
10
15
0
5
10
15
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0



1.5 Interpret scores as distances and cluster the amino acids

We'll use Pycluster to pass a distance matrix directly to Cluster3 and NetworkX to extract the leaves of the clustered tree in depth-first-search order.

(NetworkX and Pycluster can both be installed via Canopy's package manager. Pycluster can also be installed as part of the larger BioPython package).

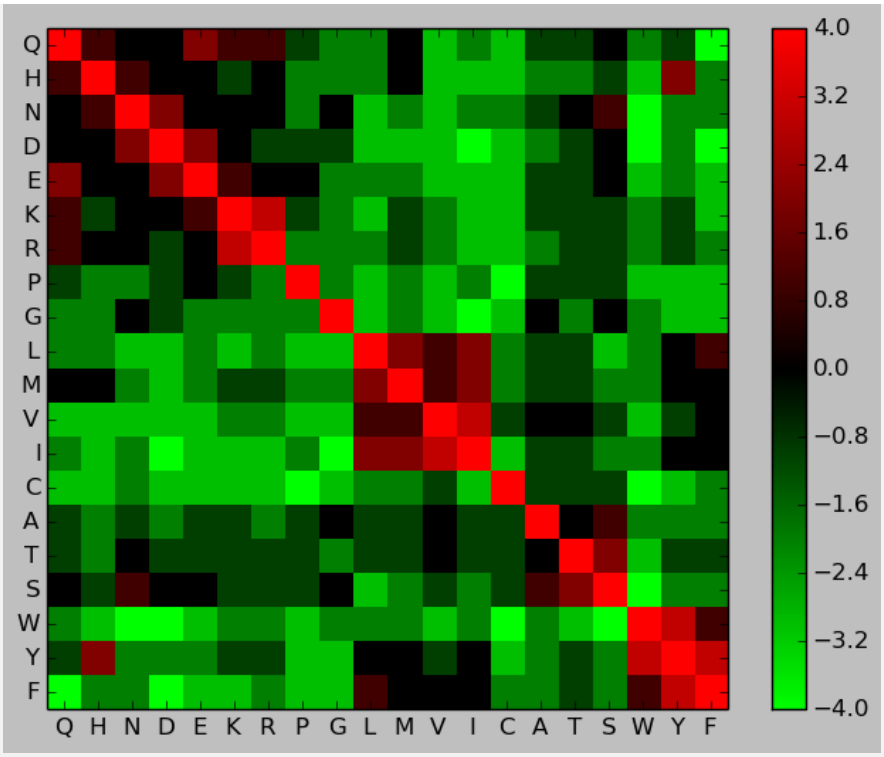
```
import networkx as nx
try:
    import Pycluster
except ImportError:
    import Bio.Cluster as Pycluster

class ScoreCluster:
    def __init__(self, S, alpha_aa = "ACDEFGHIKLMNPQRSTVWY"):
        """Initialize from numpy array of scaled log odds scores."""
        (x,y) = S.shape
        assert(x == y == len(alpha_aa))

        # Interpret the largest score as a distance of zero
        D = max(S.reshape(x**2))-S
        # Maximum-linkage clustering, with a user-supplied distance matrix
        tree = Pycluster.treecluster(distancematrix = D, method = "m")

        # Use NetworkX to read out the amino-acids in clustered order
        G = nx.DiGraph()
        for (n,i) in enumerate(tree):
```

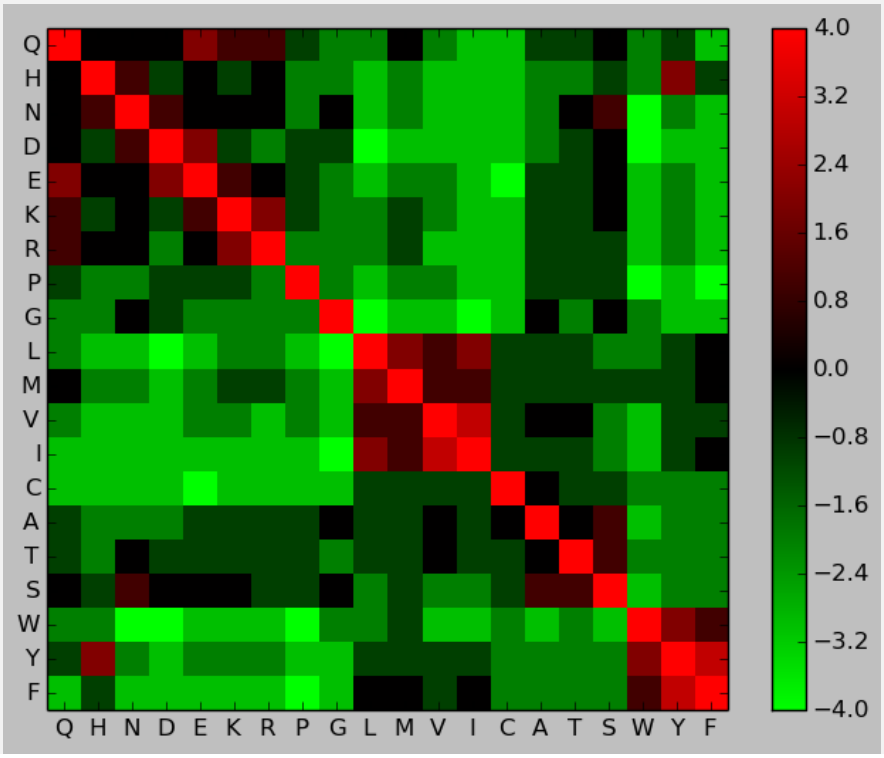
```
            for j in (i.left, i.right):
                G.add_edge(-(n+1),j)

        self.ordering = [i for i in nx.dfs_preorder(G, -len(tree)) if(i >= 0)]
        self.names = "".join(alpha_aa[i] for i in self.ordering)
        self.C = self.permute(S)

    def permute(self, S):
        """Given square matrix S in alphabetical order, return rows and columns
        of S permuted to match the clustered order."""
        return array([[S[i][j] for j in self.ordering] for i in self.ordering])
```

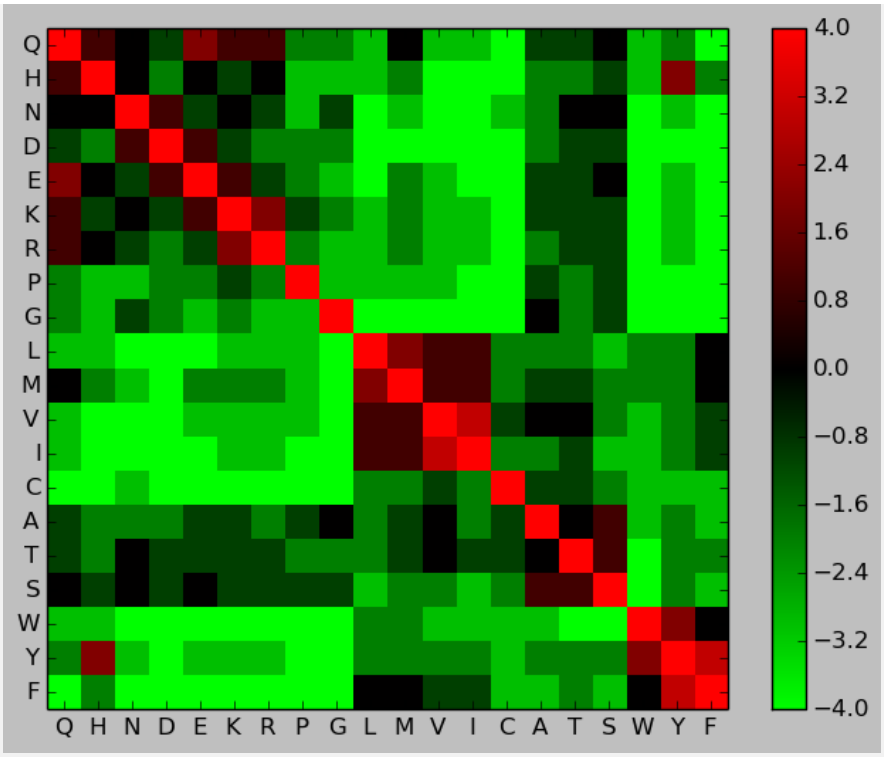
```
clustered = [ScoreCluster(i) for i in scores]
```

For comparison, choose a single clustering to apply to all three scoring matrices

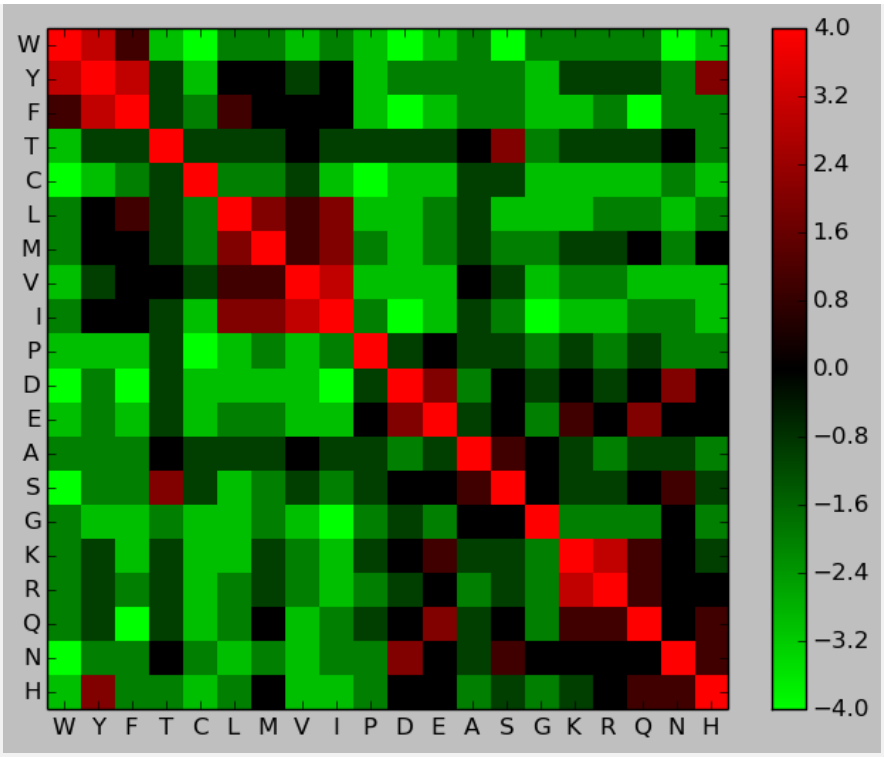
```
clust45 = [clustered[0].permute(i) for i in scores]
# Simple green->red non-overlapping gradient
cdict = {"red":((0.,0.,0.),(.5,0.,0.),(1.,1.,1.)),
         "green":((0.,1.,1.),(.5,0.,0.),(1.,0.,0.)),
         "blue":((0.,0.,0.),(.5,0.,0.),(1.,0.,0.))}
# map gradient to 256 actual RGBA values
myc = matplotlib.colors.LinearSegmentedColormap("myc",cdict,256)
for i in clust45:
    fig = figure()
    # expand color gradient from (0,1) to (-4,4)
    imshow(i, cmap = myc, clim = (-4,4), interpolation = "nearest")
    xticks(range(len(clustered[0].names)),clustered[0].names)
    yticks(range(len(clustered[0].names)),clustered[0].names)
    colorbar()
display(*(getfigs()[-3:]))
```

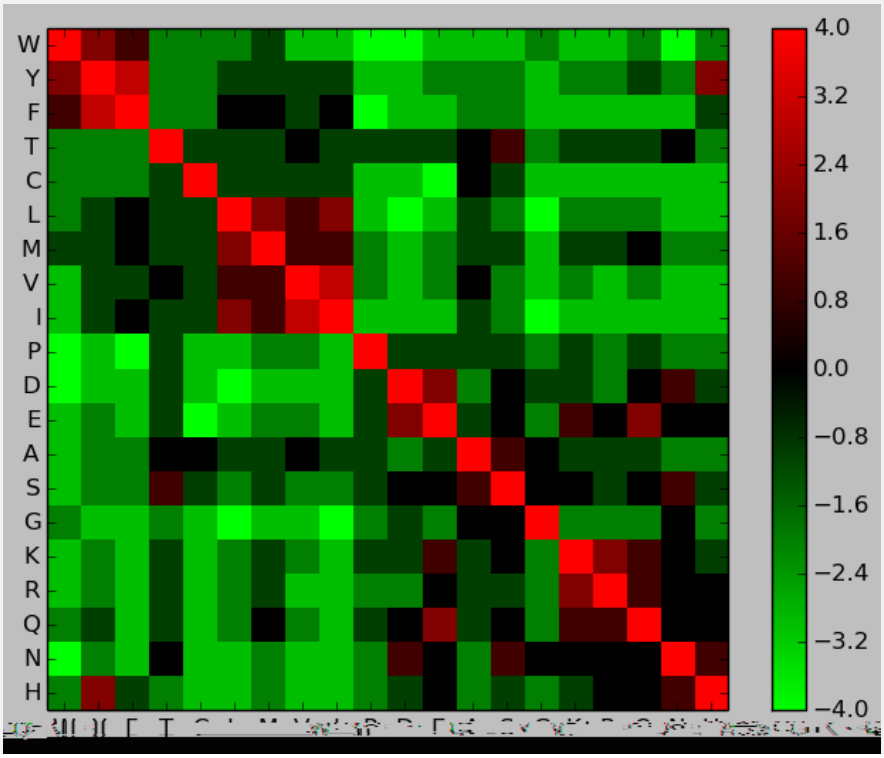

Q H N D E K R P G L M V I C A T S W Y F
Q H N D E K R P G L M V I C A T S W Y F
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0

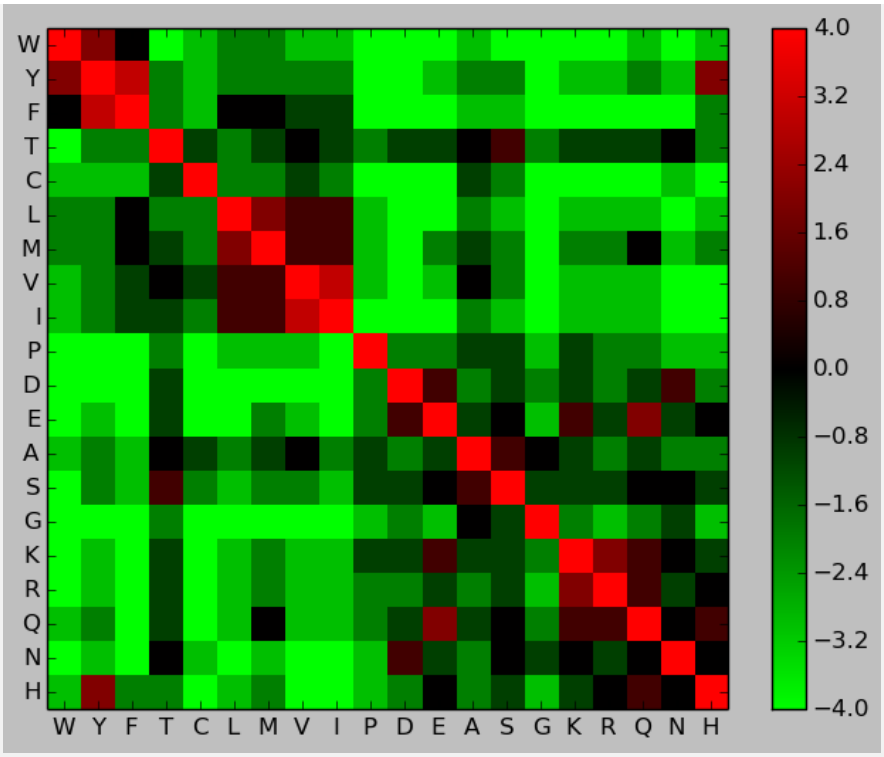

Q H N D E K R P G L M V I C A T S W Y F
Q H N D E K R P G L M V I C A T S W Y F
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0




```
clust62 = [clustered[1].permute(i) for i in scores]
# Simple green->red non-overlapping gradient
cdict = {"red":((0.,0.,0.),(.5,0.,0.),(1.,1.,1.)),
         "green":((0.,1.,1.),(.5,0.,0.),(1.,0.,0.)),
         "blue":((0.,0.,0.),(.5,0.,0.),(1.,0.,0.))}
# map gradient to 256 actual RGBA values
myc = matplotlib.colors.LinearSegmentedColormap("myc",cdict,256)
for i in clust62:
    fig = figure()
    # expand color gradient from (0,1) to (-4,4)
    imshow(i, cmap = myc, clim = (-4,4), interpolation = "nearest")
    xticks(range(len(clustered[1].names)),clustered[1].names)
    yticks(range(len(clustered[1].names)),clustered[1].names)
    colorbar()
display(*(getfigs()[-3:]))
```


W
Y
F
T
C
L
M
V
I
P
D
E
A
S
G
K
R
Q
N
H
W Y F T C L M V I P D E A S G K R Q N H
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0


W
Y
F
T
C
L
M
V
I
P
D
E
A
S
G
K
R
Q
N
H
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0


W
Y
F
T
C
L
M
V
I
P
D
E
A
S
G
K
R
Q
N
H
W Y F T C L M V I P D E A S G K R Q N H
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0