1 Practical Bioinformatics { Day 7

1.1 Homework: ungapped alignment

Michael's solution to the homework problem:

```
def offset(refSeq2, testSeq2):
    '''Determines the alignment score, comparing the testSeq to refSeq'''
    # Initialize posScore at minimal value
    maxScore = len(testSeq2)*-1
    bestPos = 0

    # Extend refSeq to the maximum possible length that testSeq could occupy
    extendRef = "."*len(testSeq2)+refSeq2+"."*len(testSeq2)

    # Loop over all possible "first base" positions in extendRef
    for pos in range(len(extendRef)-len(testSeq2)):
        # Defines a fragement of the refSeq to feed into alignScore
        fragment = extendRef[pos:pos+len(testSeq2)]
        posScore = alignScore(fragment,testSeq2)
        if (posScore > maxScore):
            maxScore = posScore
            bestPos = pos

    # Corrects for extending the refSeq at the beginning
    bestPos = bestPos-len(testSeq2)

    return bestPos
```

1.2 Dotplots

For memory-efficient, easy to navigate dotplots, use DOTTER (or pydotter on OS X). Here's a quick demo of hacking a dotplot with scipy's convolution function

Pasting in *Saccharomyces* RNA Pol II as an example sequence, using the re (regular expression) module to remove whitespace (*c.f.* http://docs.python.org/library/re.html for details).

```
import re
pol2_seq = re.sub("[\s]+","","""
    MVGQQYSSAPLRTVKEVQFGLFSPEEVRAISVAKIRFPETMDETQTRAKIGGLNDPRLGSIDRNLKCQTC
QEGMNECPGHFGHIDLAKPVFHVGFIAKIKKVCECVCMHCGKLLLDEHNELMRQALAIKDSKKRFAAIWT
LCKTKMVCETDVPSEDDPTQLVSRGGCGNTQPTIRKDGLKLVGSWKKDRATGDADEPELRVLSTEEILNI
FKHISVKDFTSLGFNEVFSRPEWMILTCLPVPPPPVRPSISFNESQRGEDDLTFKLADILKANISLETLE
HNGAPHHAIEEAESLLQFHVATYMDNDIAGQPQALQKSGRPVKSIRARLKGKEGRIRGNLMGKRVDFSAR
TVISGDPNLELDQVGVPKSIAKTLTYPEVVTPYNIDRLTQLVRNGPNEHPGAKYVIRDSGDRIDLRYSKR
AGDIQLQYGWKVERHIMDNDPVLFNRQPSLHKMSMMAHRVKVIPYSTFRLNLSVTSPYNADFDGDEMNLH
VPQSEETRAELSQLCAVPLQIVSPQSNKPCMGIVQDTLCGIRKLTLRDTFIELDQVLNMLYWVPDWDGVI
PTPAIIKPKPLWSGKQILSVAIPNGIHLQRFDEGTTLLSPKDNGMLIIDGQIIFGVVEKKTVGSSNGGLI
HVVTREKGPQVCAKLFGNIQKVVNFWLLHNGFSTGIGDTIADGPTMREITETIAEAKKKVLDVTKEAQAN
LLTAKHGMTLRESFEDNVVRFLNEARDKAGRLAEVNLKDLNNVKQMVMAGSKGSFINIAQMSACVGQQSV
EGKRIAFGFVDRTLPHFSKDDYSPESKGFVENSYLRGLTPQEFFFHAMGGREGLIDTAVKTAETGYIQRR
LVKALEDIMVHYDNTTRNSLGNVIQFIYGEDGMDAAHIEKQSLDTIGGSDAAFEKRYRVDLLNTDHTLDP
```

```
SLLESGSEILGDLKLQVLLDEEYKQLVKDRKFLREVFVDGEANWPLPVNIRRIIQNAQQTFHIDHTKPSD
LTIKDIVLGVKDLQENLLVLRGKNEIIQNAQRDAVTLFCCLLRSRLATRRVLQEYRLTKQAFDWVLSNIE
AQFLRSVVHPGEMVGVLAAQSIGEPATQMTLNTFHFAGVASKKVTSGVPRLKEILNVAKNMKTPSLTVYL
EPGHAADQEQAKLIRSAIEHTTLKSVTIASEIYYDPDPRSTVIPEDEEIIQLHFSLLDEEAEQSFDQQSP
WLLRLELDRAAMNDKDLTMGQVGERIKQTFKNDLFVIWSEDNDEKLIIRCRVVRPKSLDAETEAEEDHML
KKIENTMLENITLRGVENIERVVMMKYDRKVPSPTGEYVKEPEWVLETDGVNLSEVMTVPGIDPTRIYTN
SFIDIMEVLGIEAGRAALYKEVYNVIASDGSYVNYRHMALLVDVMTTQGGLTSVTRHGFNRSNTGALMRC
SFEETVEILFEAGASAELDDCRGVSENVILGQMAPIGTGAFDVMIDEESLVKYMPEQKITEIEDGQDGGV
TPYSNESGLVNADLDVKDELMFSPLVDSGSNDAMAGGFTAYGGADYGEATSPFGAYGEAPTSPGFGVSSP
GFSPTSPTYSPTSPAYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSP
SYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPSYSPTSPAYSPTSPSYSPTSPSYSPTSP
SYSPTSPSYSPTSPNYSPTSPSYSPTSPGYSPGSPAYSPKQDEQKHNENENSR""")
```

```
pol2_seq
```

```
'MVGQQQYSSAPLRTVKEVQFGLFSPEEVRAISVAKIRFPETMDETQTRAKIGGLNDPRLGSIDRNLKCQTCQEGMNECPGHFGHIDLAKPVFHVGFIAKIKKVC
```

Forming a simple identity dotplot as a numpy array

```
pol2 = array([[i == j for j in pol2_seq] for i in pol2_seq])
```

Alternatively, we could use the BLOSUM62 matrix to generate a dotplot with knowledge of amino-acid similarity, *e.g.*:

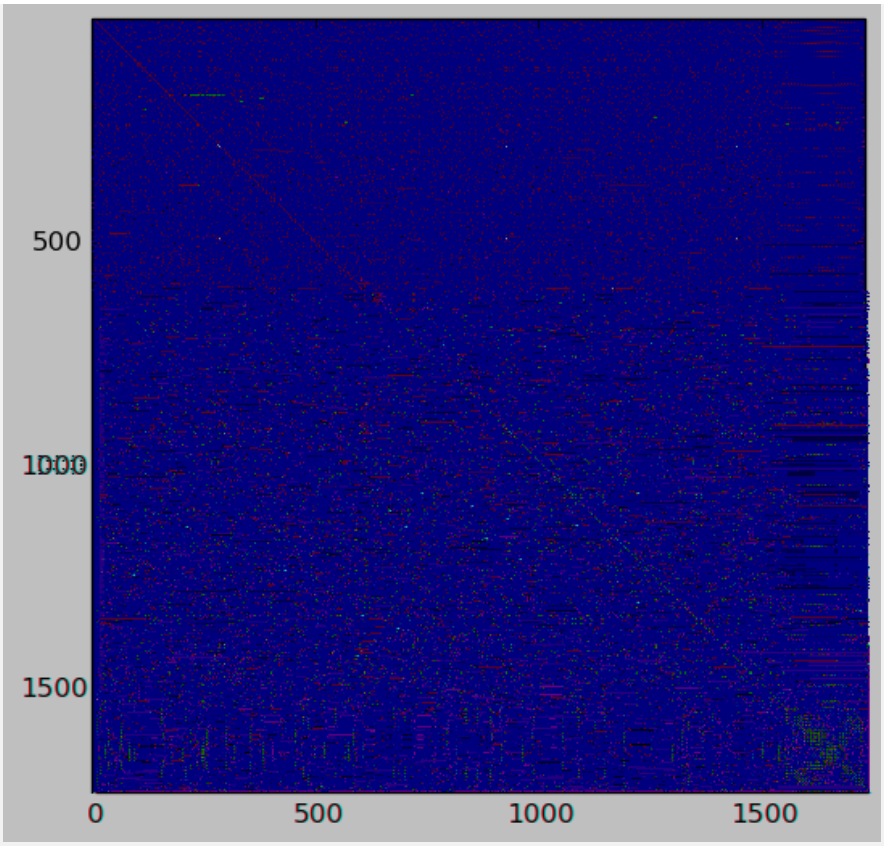
```
from blosum62 import blosum62
pol2_blosum62 = array([[blosum62[i][j] for j in pol2_seq] for i in pol2_seq])
```

Matplotlib's imshow plots a two-dimensional array as a heatmap. Here, we use it to plot our identity-based dotplot of pol2:

```
fig = figure()
imshow(pol2)
```

```
<matplotlib.image.AxesImage at 0x3fe9e90>
```

```
display(fig)
```

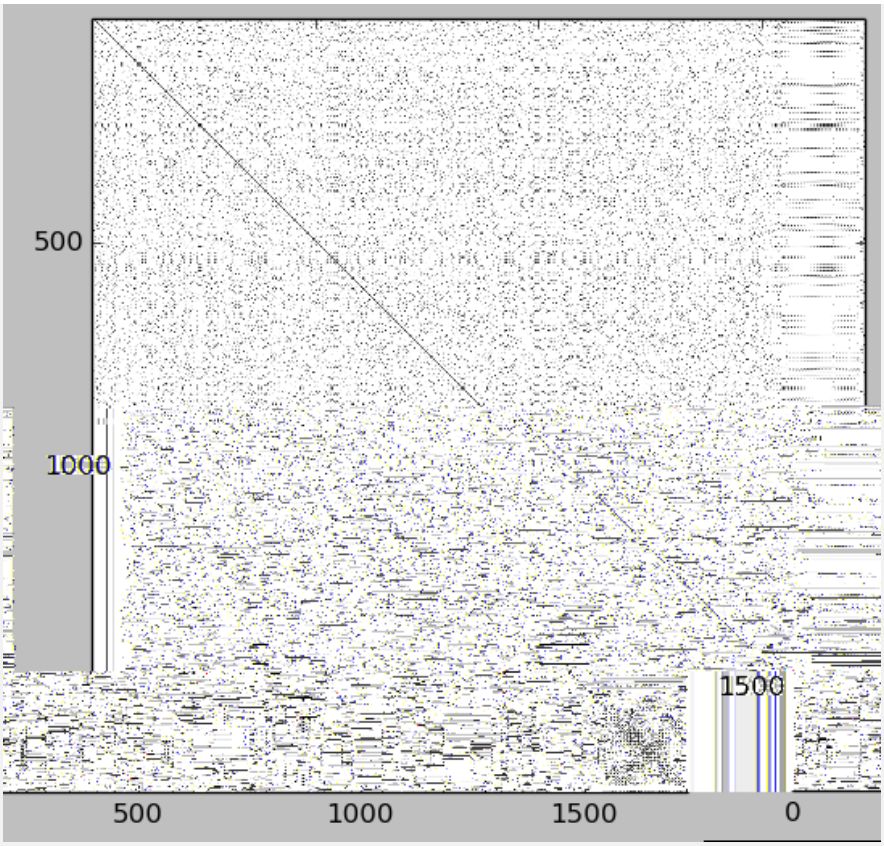


Switching to greyscale for better contrast:

```
fig = figure()
imshow(pol2, cmap = "Greys")
```

```
<matplotlib.image.AxesImage at 0x6032cd0>
```

```
display(fig)
```



DOTTER uses a windowed average on the diagonals of the matrix to remove noise. Here, we approximate DOTTER's averaging by convolving a simple diagonal mask with our matrix. See sections 12.0 and 13.1 of Numerical Recipes if you are interested in the details of convolution.

Create our mask:

```
mask = identity(25)
```

```
mask
```

```
array([[ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
```

```
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]])
```

Do the convolution:

```
import scipy.signal
```

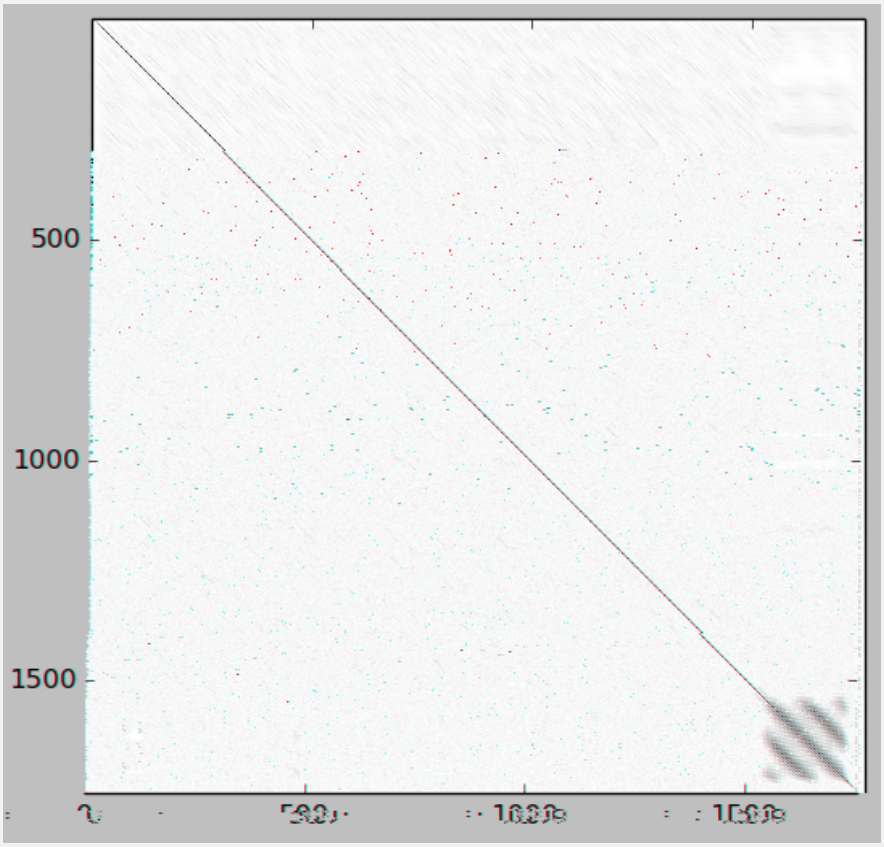
```
C = scipy.signal.fftconvolve(pol2, mask)
```

Plot the smoothed dotplot in greyscale:

```
fig = figure()
imshow(C, cmap = "Greys")
```

```
<matplotlib.image.AxesImage at 0x8083d90>
```

```
display(fig)
```

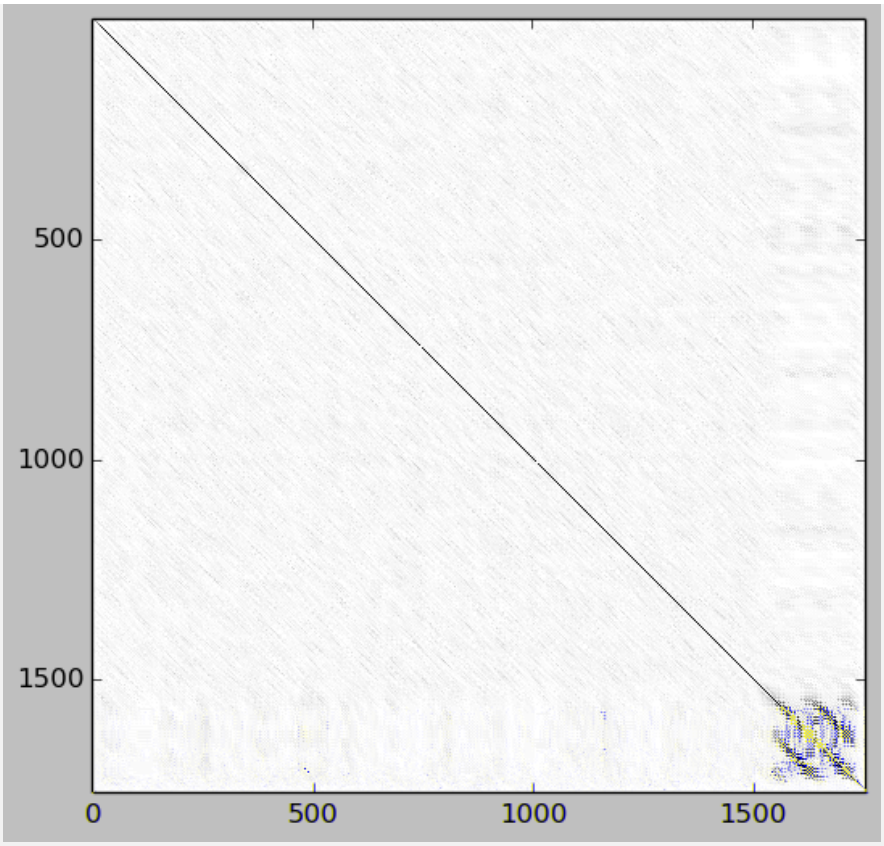


By default, imshow uses gaussian blur for anti-aliasing. This is nice for natural images, but not what we want when visualizing a matrix. Here, we set *interpolation="nearest"* to render the elements of our matrix as simple rectangles:

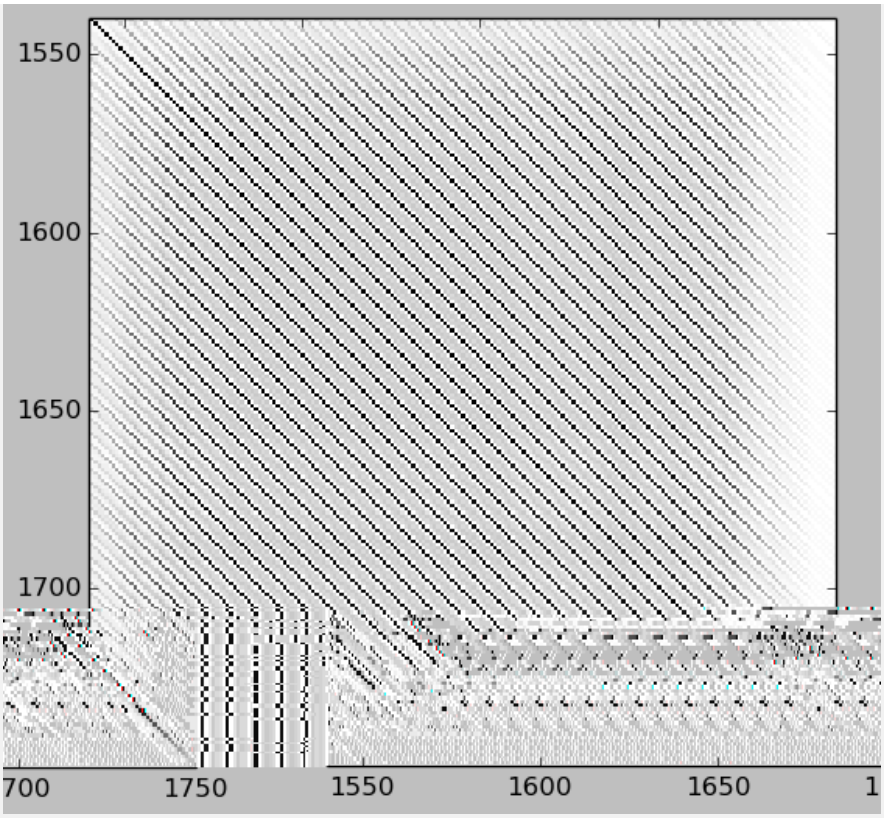
```
fig = figure()
imshow(C, cmap = "Greys", interpolation = "nearest")
```

```
<matplotlib.image.AxesImage at 0x9a6da10>
```

```
display(fig)
```



And zooming in on the C-terminal heptameoa3 ep3(zo)eat:



1.3 Other plotting

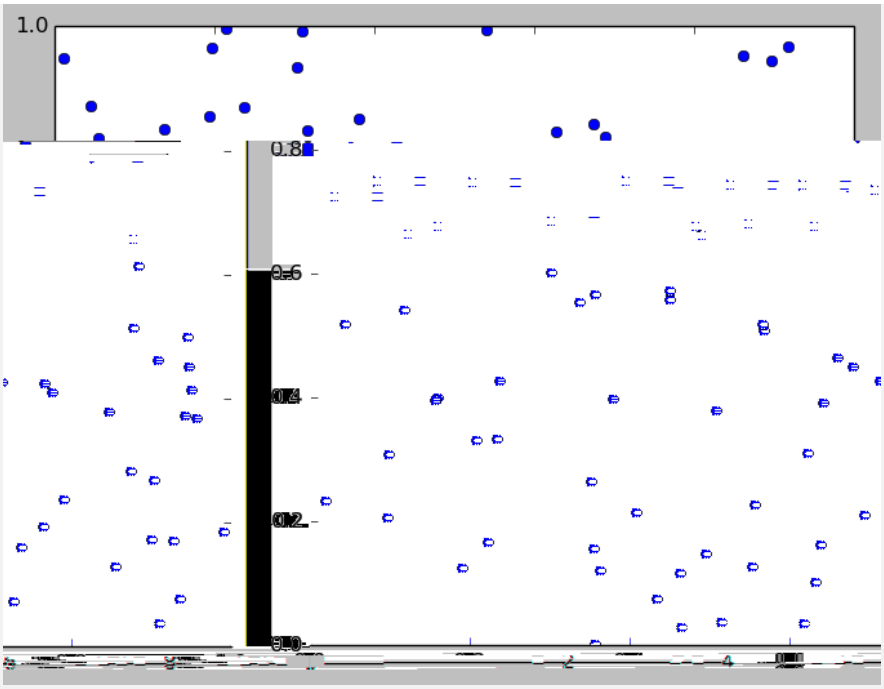
Here's how to use the matplotlib plotting interface for simple scatter plots of two equal length vectors:

Create two random numpy arrays of 100 uniformly distributed values from 0 to 1 and scatter plot them as blue ("b") circles ("o"):

```
x = rand(100)
y = rand(100)
fig = figure()
plot(x,y,"bo")
```

```
[<matplotlib.lines.Line2D at 0x98e2210>]
```

```
display(fig)
```



See http://matplotlib.org/gallery.html for more matplotlib plotting examples.

```
%logstart -o BMS270b.2013.07.log
```

```
Activating auto-logging. Current session state plus future input saved.
Filename       : BMS270b.2013.07.log
Mode           : backup
Output logging : True
Raw input log  : False
Timestamping   : False
State          : active
```