# 2018-06-27-New_data_analysis

June 27, 2018

```
In [1]: %matplotlib inline
        %load_ext autoreload
        %autoreload 2
```

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import mybiotools as mbt
        import hpiptools as ht
```

## 1 2018-06-27 New data analysis

We now have a complete list of barcodes that we can trust that were really good integrations. Let's have a quick look at some basic aspects of the integrations.

```
In [3]: # load data
        hpip_root = '/home/rcortini/work/CRG/projects/hpip'
        hpip_integrations_fname = '%s/data/hpip-integrations.txt'%(hpip_root)
```

First things first: now we have a list of promoters that are possible candidates for each barcode. Let's write a parser of the file that takes this into account. First, let's see how the number of retrieved barcodes varies as a function of how strict we are with promoter assignment.

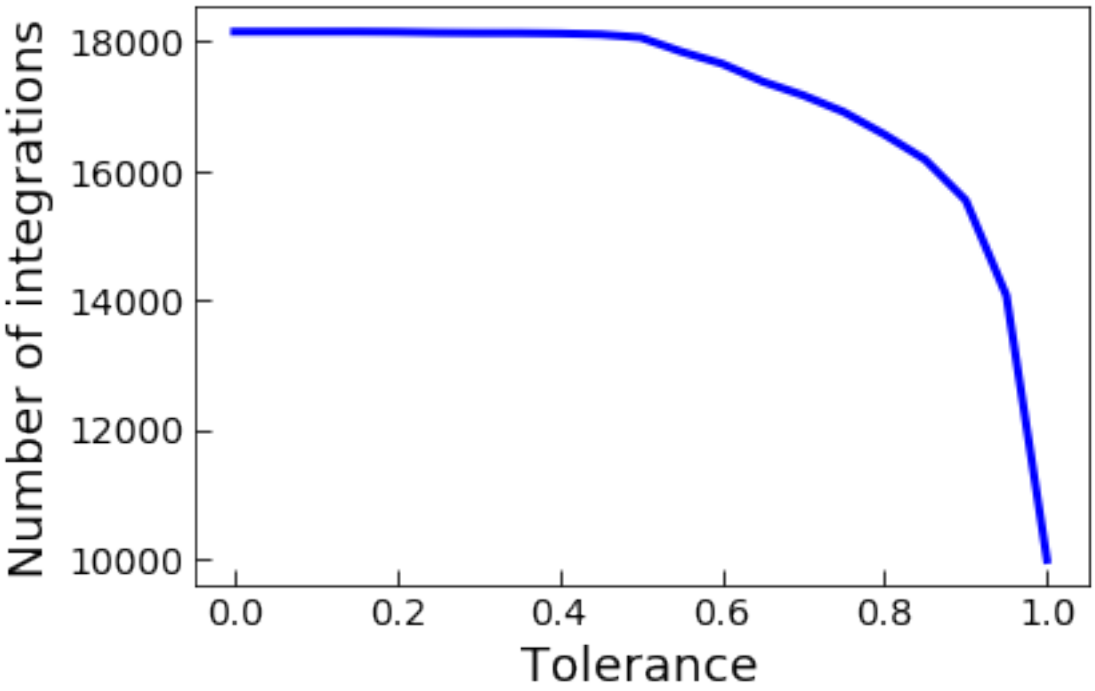
```
In [14]: def load_hpip_integrations(fname, tol) :
             """
             Load the HPIP integrations file `fname`. For each of the barcodes,
             we accept only the promoter that has a probability of being called
             greater than `tol`. If none passes the test, then we throw the barcode
             away.
             """
             hpip_dtype = [
                 ('barcode','S32'),
                 ('chr','S32'),
                 ('pos',np.int32),
                 ('strand','S2'),
                 ('iPCR',np.int32),
                 ('cDNA',np.int32),
                 ('gDNA',np.int32),
```

```
            ('lib','S8'),
            ('rep','S8'),
            ('promoter','S32'),
            ('p',np.float32)
        ]
        with open(fname, 'r') as f :
            bcds = []
            for line in f :
                curatedline = line.strip('\n').split('\t')
                bcd, chrom, pos, strand, \
                lib, rep, iPCR, cDNA, gDNA = curatedline[:9]
                proms_raw = curatedline[9:]
                f = lambda x : float(x.split(':')[1])
                ps = [f(p) for p in proms_raw]
                maxp = max(ps)
                if maxp < tol :
                    continue
                else :
                    imax = ps.index(maxp)
                prom, p = proms_raw[imax].split(':')
                bcds.append((bcd, chrom, pos, strand, iPCR, cDNA, gDNA, lib, rep, prom, p))
        return np.array(bcds, dtype=np.dtype(hpip_dtype))
```

So now let's see how the number of integrations varies as a function of the tolerance.

```
In [28]: tols = np.arange(0.,1.05,0.05)
         size_tols = np.zeros(tols.size)
         for i, tol in enumerate(tols) :
             hpip = load_hpip_integrations(hpip_integrations_fname, tol)
             size_tols[i] = hpip.size
```

```
In [31]: plt.plot(tols, size_tols, linewidth=3)
         plt.xlabel('Tolerance')
         plt.ylabel('Number of integrations')
         plt.show()
```




Okay this is the kind of expected behaviour. There are a lot of integrations that are thrown away between 80% and 100% of tolerance. Let's choose 90% tolerance as a measure, for now.

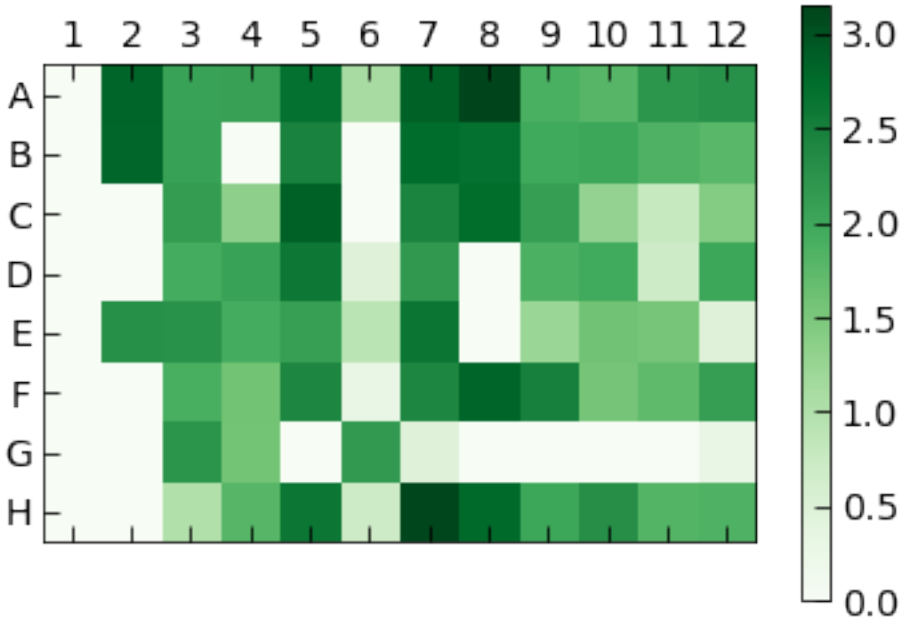
```
In [32]: tol = 0.9
         hpip = load_hpip_integrations(hpip_integrations_fname, tol)
```

Let's now ask another basic question: how many integrations do we have per promoter?

```
In [33]: prom_libs = range(1,13)
         prom_lib_idx = {str(prom_lib) : prom_lib-1 for prom_lib in prom_libs}
         prom_class_idx = {'A':0,'B':1,'C':2,'D':3,
                           'E':4,'F':5,'G':6,'H':7}
```

```
In [36]: promoters = np.zeros((8,12))
         for integration in hpip :
             prom_class, prom_lib = ht.prom_id(integration['promoter'])
             promoters[prom_class_idx[prom_class], prom_lib_idx[prom_lib]] += 1
```

```
In [37]: cax = plt.matshow(np.log10(promoters+1), cmap=plt.cm.Greens)
         plt.xticks(prom_lib_idx.values(), prom_lib_idx.keys())
         plt.yticks(prom_class_idx.values(), prom_class_idx.keys())
         plt.colorbar(cax)
         plt.show()
```

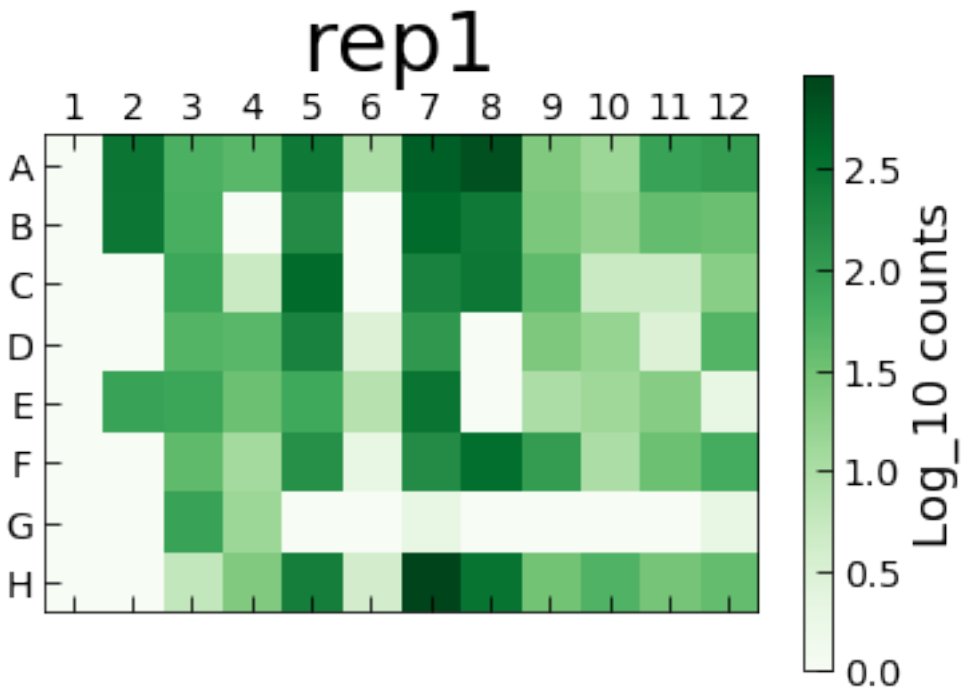


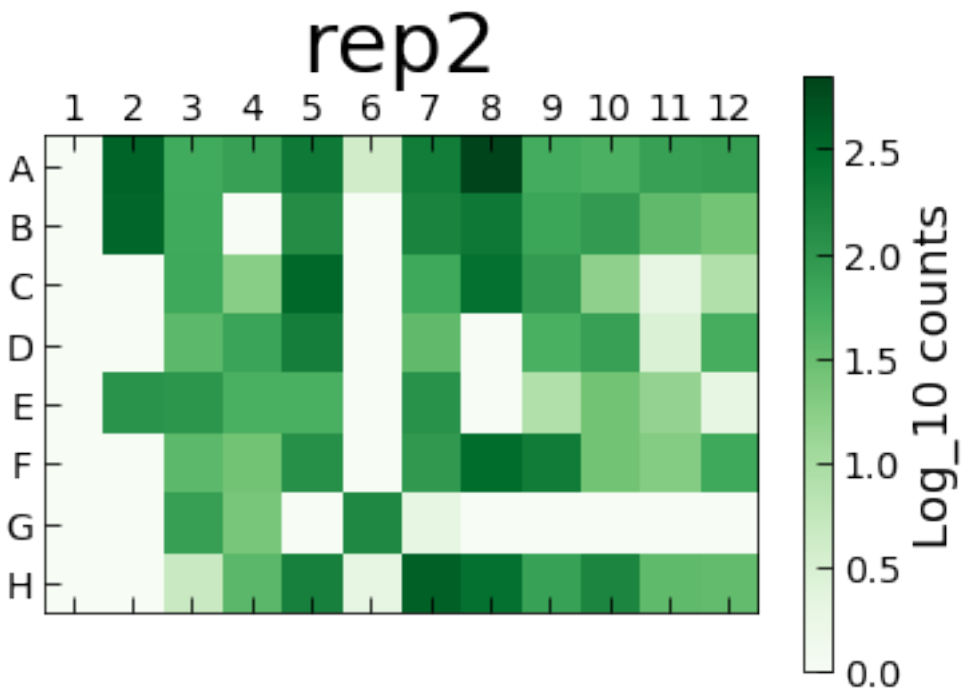

Let's separate the results by replicate.

```
In [42]: reps = np.unique(hpip['rep'])
         hpip_rep = {}
         for rep in reps :
             hpip_rep[rep] = np.array([i for i in hpip if i['rep']==rep])

In [45]: promoters_rep = {}
         for rep in reps :
             promoters_rep[rep] = np.zeros((8,12))
             for integration in hpip_rep[rep] :
                 prom_class, prom_lib = ht.prom_id(integration['promoter'])
                 promoters_rep[rep][prom_class_idx[prom_class], prom_lib_idx[prom_lib]] += 1

In [49]: for rep in reps :
             cax = plt.matshow(np.log10(promoters_rep[rep]+1), cmap=plt.cm.Greens)
             plt.xticks(prom_lib_idx.values(), prom_lib_idx.keys())
             plt.yticks(prom_class_idx.values(), prom_class_idx.keys())
             cbar = plt.colorbar(cax)
             plt.title(rep, y=1.1, fontsize=32)
             cbar.set_label("Log_10 counts")
             plt.show()
```




Okay, this is the stop signal. The numbers are insufficient to do anything else.