Random commutators in symmetric groups

June 22, 2018

1 Commutators

We study the distribution of a commutator $c = rur^{-1}u^{-1}$ when both r and u are taken at random in some symmetric group S_n with n large. Note that $c \in A_n$, the alternating group. We compare the distribution of c with the uniform distribution in A_n looking at two statistics:

- the number of fixed points
- the number of cycles

1.1 Empirical distributions of commutators

The first approach to generate random permutations is to use `SymmetricGroup(n)`. However this is dramatically slow...

```
In [1]: %%time
        n = 5
        S = SymmetricGroup(5)
        for _ in range(1000):
            r = S.random_element()   # pick r at random
            u = S.random_element()   # pick u at random
            c = r*u*r^-1*u^-1        # commutator
            t = c.cycle_tuples()     # decomposition in cycles

CPU times: user 2.45 s, sys: 793 ms, total: 3.24 s
Wall time: 3.5 s
```

Instead we use the functions `perm_compose`, `perm_compose_i`, `perm_cycle_tuples` that belongs to the package `surface_dynamics`. These functions operate on lists of integers in $\{0, 1, \ldots, n-1\}$.

```
In [2]: from surface_dynamics.misc.permutation import perm_compose, perm_compose_i, perm_cycle

        def sample(n, size):
            r"""
            Return the distribution of number of fixed points and number of cycles
            for a random commutator.
```

```
            INPUT:

            - ``n`` - rank of the symmetric group

            - ``size`` - size of the sample
            """
            r = range(n)
            u = range(n)
            fps = [0] * (n+1)   # number of fixed points
            ts  = [0] * (n+1)   # number of cycles
            for _ in range(size):
                shuffle(r)  # randomly mix r
                shuffle(u)  # randomly mix u
                c = perm_compose(perm_compose(r, u), perm_compose_i(r, u))  # commutator

                # count fixed points
                fp = sum(c[i] == i for i in range(n))
                fps[fp] += 1

                # count number of cycles (~ genus)
                t = len(perm_cycle_tuples(c,True))
                ts[t] += 1

            return (fps, ts)
```

```
In [3]: sample(5, 10)
```

```
Out[3]: ([3, 2, 5, 0, 0, 0], [0, 3, 0, 7, 0, 0])
```

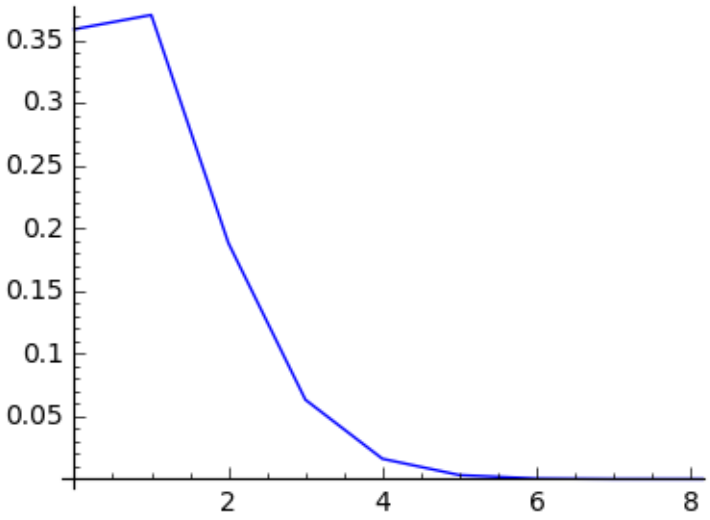
```
In [4]: # Fix the size of the symmetric group and the size of the sample
        n = 50
        size = 50000
```

```
In [5]: %%time
        fps, ncycs = sample(n, size)
```

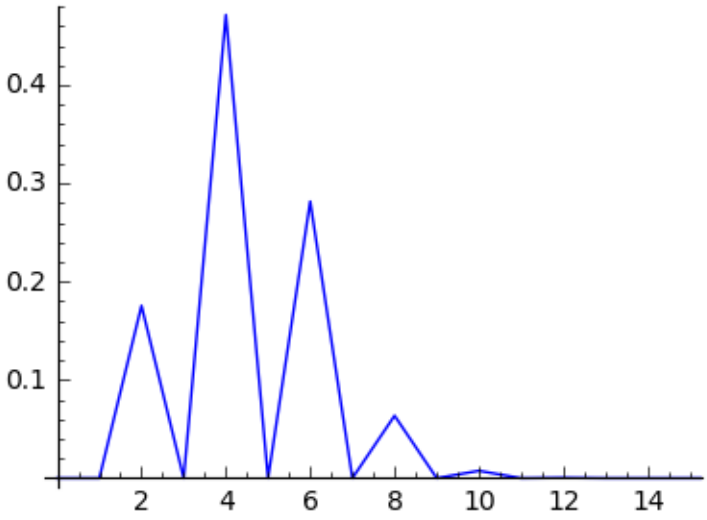
```
CPU times: user 3.23 s, sys: 293 µs, total: 3.23 s
Wall time: 3.23 s
```

```
In [6]: # renormalize our samples
        for i in range(n+1):
            fps[i] /= size
            ncycs[i] /= size
```

```
In [7]: L = list_plot(fps, plotjoined=True, color='blue')
        L.show(xmax=8, figsize=4)
```

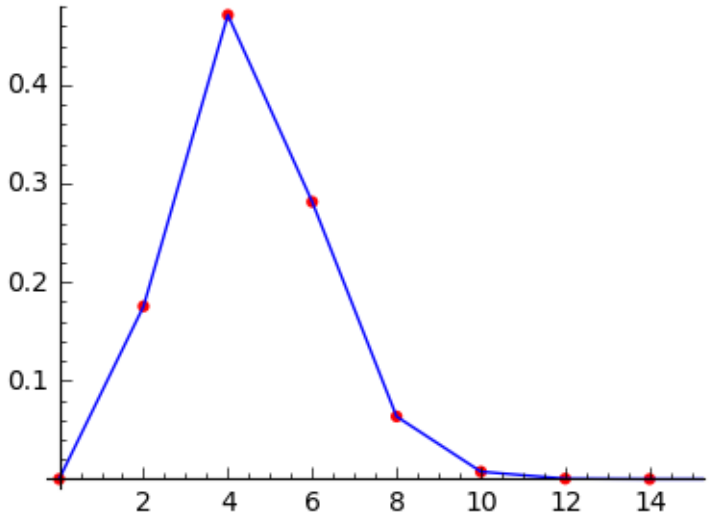



```
In [8]: L = list_plot(ncycs, plotjoined=True, color='blue')
        L.show(xmax=15, figsize=4)
```



One can remark that the number of cycles is always an even number... this is because a commutator belongs to the alternating group. To get a nicer graph, we can plot only the values at even points.

```
In [9]: ncycs2 = [(i,ncycs[i]) for i in range(0,n+1,2)]
        G = line2d(ncycs2, color='blue') + point2d(ncycs2, color='red', pointsize=20)
        G.show(xmax=15, figsize=4)
```



1.2 Exact distributions in A_n

Now we want to compare our empirical distribution of commutatoris with the exact distribution in the alternating group A_n. For that purpose, we list partitions (that correspond to conjugacy classes of permutations). Note that for $n = 50$ we already have 204226 partitions and 102162 of them are even and corresponds to conjugacy classes in A_n.

```
In [10]: P = Partitions(n)
         print P.cardinality()
         P0 = [p for p in P if p.sign() == 1]
         print len(P0)
```

```
204226
102162
```

```
In [11]: %%time
         # small check: the sum of conjugacy class sizes should be the cardinality of An
         sum(p.conjugacy_class_size() for p in P0) == factorial(n) / 2
```

```
CPU times: user 1.36 s, sys: 70.9 ms, total: 1.43 s
Wall time: 1.35 s
```

```
Out[11]: True
```

```
In [12]: %%time
         # now compute distributions
         distrib_fp = [0] * (n+1)
         distrib_ncyc  = [0] * (n+1)
```

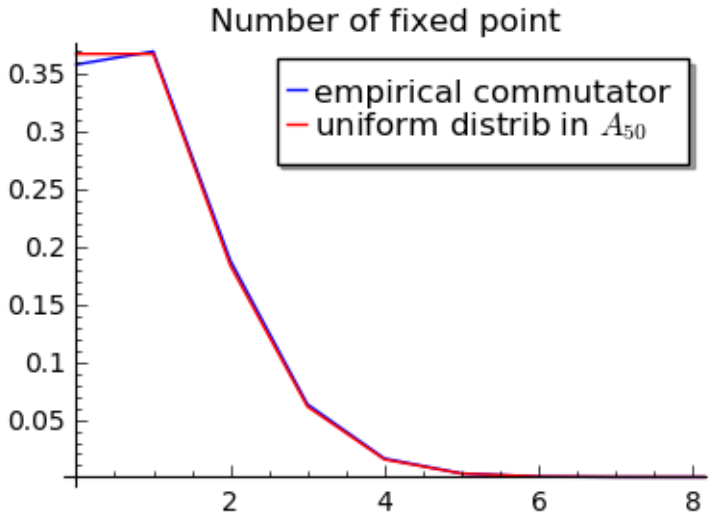
```
        for p in P:
            if p.sign() == -1:
                continue

            distrib_fp[p._list.count(1)] += p.conjugacy_class_size()
            distrib_ncyc[len(p._list)] += p.conjugacy_class_size()
        for i in range(n+1):
            distrib_fp[i] *= 2/factorial(n)
            distrib_ncyc[i] *= 2/factorial(n)
```

```
CPU times: user 5.56 s, sys: 52.2 ms, total: 5.62 s
Wall time: 5.56 s
```

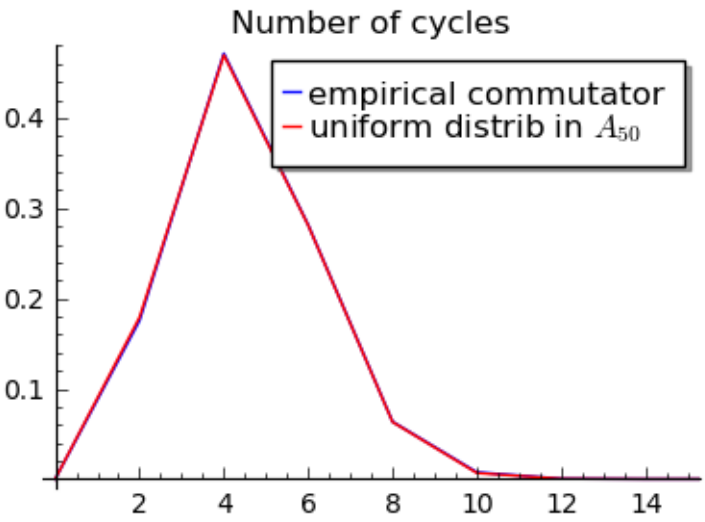
```
In [13]: L1 = list_plot(fps, plotjoined=True, color='blue', legend_label='empirical commutator
         L2 = list_plot(distrib_fp, plotjoined=True, color='red', legend_label='uniform distri
```

```
In [14]: (L1 + L2).show(title='Number of fixed point', xmax=8, figsize=4)
```




```
In [15]: # for number of cycles, we only consider even sizes
         ncycs2 = [(i,ncycs[i]) for i in range(0,n+1,2)]
         distrib_ncyc2 = [(i,distrib_ncyc[i]) for i in range(0,n+1,2)]
         L3 = line2d(ncycs2, color='blue', legend_label='empirical commutator')
         L4 = line2d(distrib_ncyc2, color='red', legend_label='uniform distrib in $A_{%d}$' % n
```

```
In [16]: (L3 + L4).show(title='Number of cycles', xmax=15, figsize=4)
```


Number of cycles
empirical commutator
uniform distrib in A_{50}
0.4
0.3
0.2
0.1
2
4
6
8
10
12
14