fast-modsym-notebook

May 11, 2017

0.0.1 Example of loading fast modular symbols code in a Jupyter notebook

Exacty this same code should work in a .py file...

```
In [2]: load("modular_symbol_map.pyx")
```

```
Compiling ./modular_symbol_map.pyx...
```

```
In [3]: A = ModularSymbols(389,sign=1).cuspidal_subspace().new_subspace().decomposition()[0]
        f = ModularSymbolMap(A)
```

```
In [4]: %timeit f._eval1(-3,7)
```

```
The slowest run took 21.72 times longer than the fastest. This could mean that an intermediate r
1000000 loops, best of 3: 1.11 ţs per loop
```

```
In [0]:
```

0.0.2 Now try $d = 29$ as in `11a.ipynb`

```
In [5]: d = 29
        # much more ms, since this code is massively faster...
        ms = [m for m in prime_range(3,100000) if gcd(m, 11) == 1 and euler_phi(m) % d == 0]
        print(ms)
```

```
[59, 233, 349, 523, 929, 1103, 1277, 1451, 1567, 1741, 1973, 2089, 2437, 2843, 3191, 3307, 3539,
```

```
In [6]: M = ModularSymbols(11,sign=1).cuspidal_submodule()
        ms_map = ModularSymbolMap(M)
        ms_denom = ZZ(ms_map.denom)
        def f(a,b):
            return ms_map._eval1(a,b)[0] / ms_denom
```

```
In [7]: f(1,11)
```

```
Out[7]: -1
```

```
In [8]: def alphas(m, d, normalize=True):
            gen = Integers(m)(primitive_root(m))
            n = euler_phi(m)//d
            b = gen^n
            h = gen^d
            if normalize:
                denom = float(sqrt(m*euler_phi(m)*log(m)))
            else:
                denom = 1
            alphas = []
            for i in range(d):
                s = 0
                for j in range(n):
                    period = f((b^i * h^j).lift(), m)
                    s += period
                alphas.append(s / denom)
            return alphas
```

```
In [9]: print alphas(ms[0], d)
```

```
[0.04232831912577409, 0.04232831912577409, 0.04232831912577409, 0.04232831912577409, 0.042328319
```

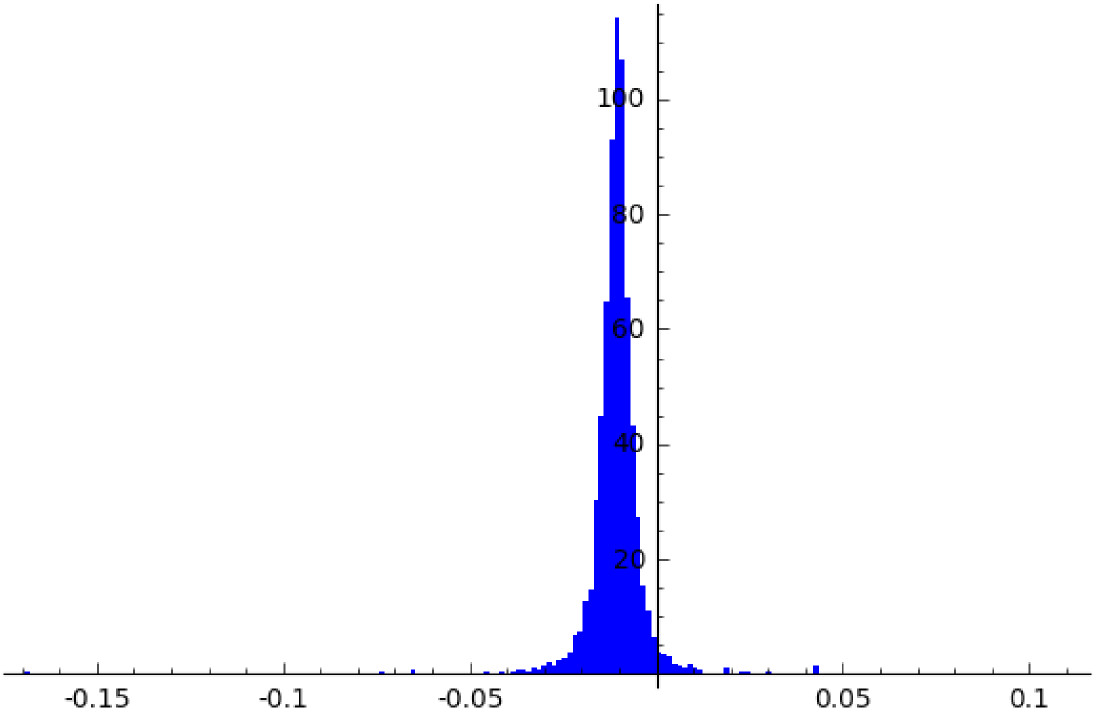
```
In [10]: %%time
         data = []
         for m in ms:
             data += alphas(m, d)
```

```
CPU times: user 1min 19s, sys: 96 ms, total: 1min 19s
Wall time: 1min 19s
```

```
In [11]: print len(data)
         t = stats.TimeSeries(data)
         print t.mean()
         t.plot_histogram(bins=200)
```

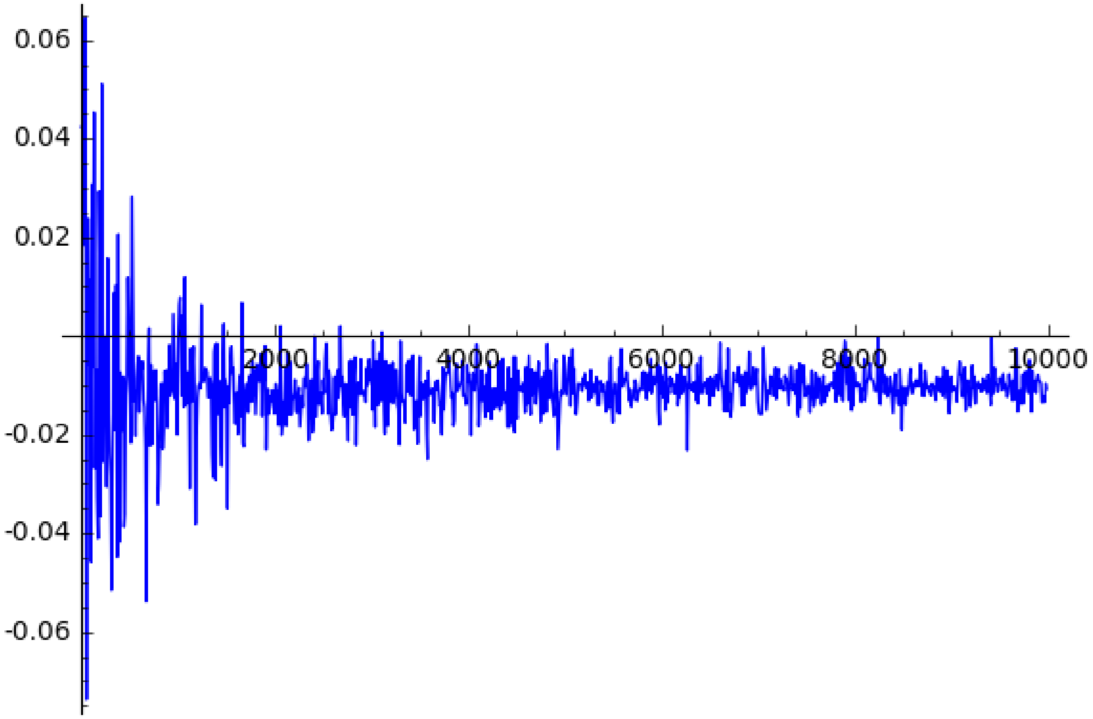
```
10005
-0.0107567619125
```

Out[11]:



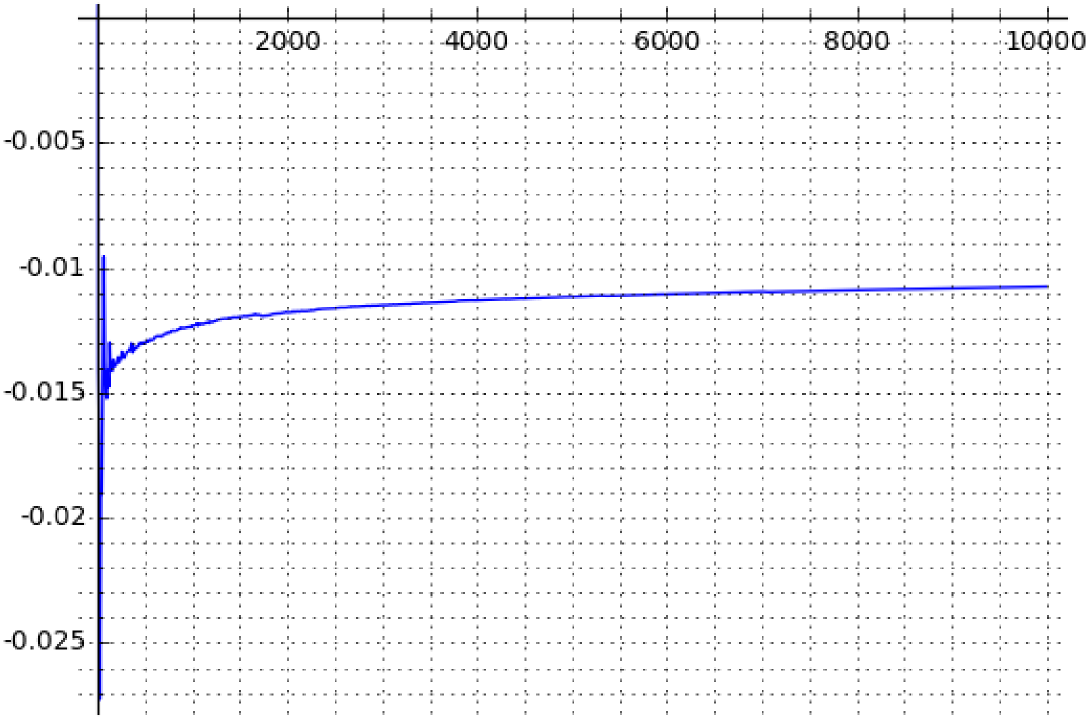
```
In [12]: t.plot()
```

```
Out[12]:
```



```
In [13]: stats.TimeSeries(t[:i].mean() for i in range(5,len(t))).plot(
                gridlines='minor', ymax=0)
```

Out[13]:



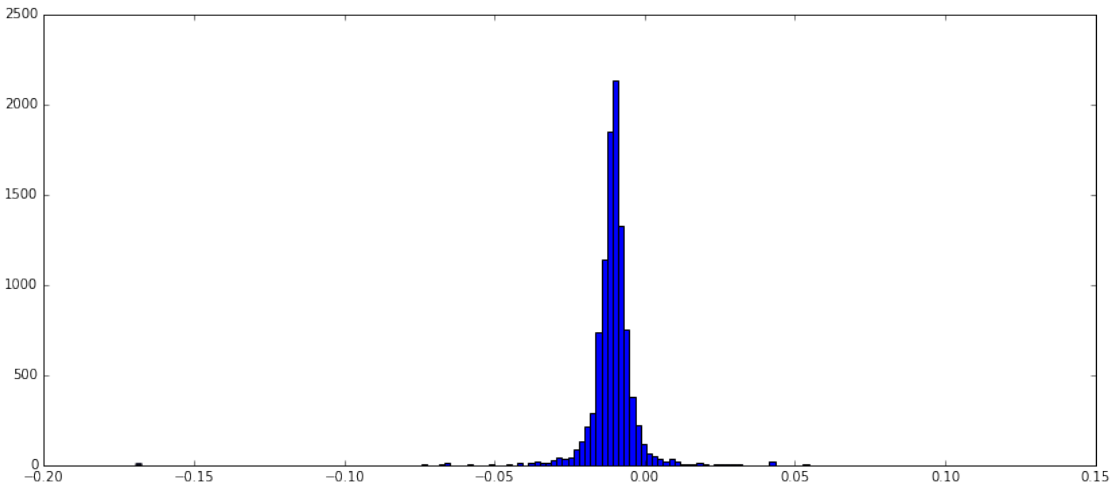
see https://docs.scipy.org/doc/scipy-0.7.x/reference/generated/scipy.stats.kurtosis.html

```
In [14]: import scipy.stats
         scipy.stats.kurtosis(t.numpy(), fisher=False)
```

```
Out[14]: 90.78104263656937
```

```
In [15]: import matplotlib.pyplot as plt
         plt.figure(figsize=(14,6))
         plt.hist(t.numpy(), bins=150)
         plt.show()
```

Out[15]:



0.0.3 Now try $d = 97$

```
In [16]: d = 97
         # much more ms, since this code is massively faster...
         ms = [m for m in prime_range(3,100000) if gcd(m, 11) == 1 and euler_phi(m) % d == 0]
         print(ms)
```

```
[389, 971, 1553, 1747, 3299, 3881, 4463, 4657, 5821, 6791, 8537, 8731, 10477, 11059, 11447, 1261
```

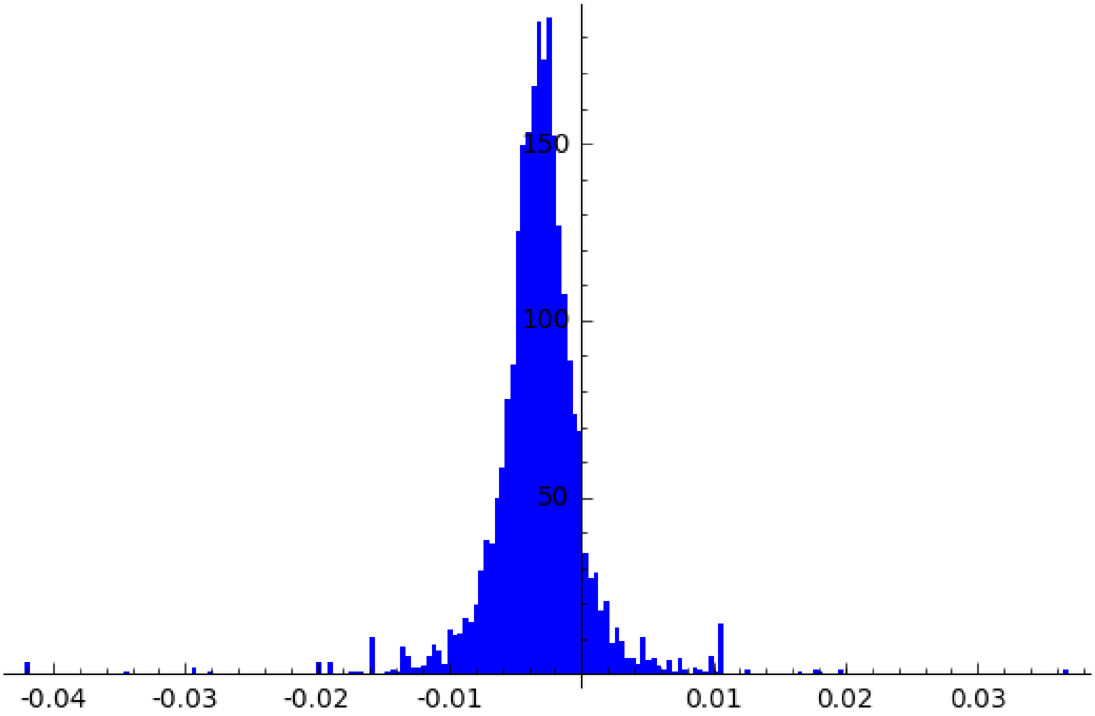
```
In [17]: %%time
         data = []
         for m in ms:
             data += alphas(m, d)
```

```
CPU times: user 25.4 s, sys: 32 ms, total: 25.4 s
Wall time: 25.4 s
```

```
In [18]: print len(data)
         t = stats.TimeSeries(data)
         print t.mean()
         t.plot_histogram(bins=200)
```

```
9603
-0.00321230474805
```

```
Out[18]:
```



```
In [19]: t.variance()

Out[19]: 1.6244063751141795e-05

In [20]: save(t, 't-11a-97')
```

0.0.4 Now try $d = 997$

```
In [21]: d = 997
         # much more ms, since this code is massively faster...
         ms = [m for m in prime_range(3,100000) if gcd(m, 11) == 1 and euler_phi(m) % d == 0]
         print(ms)

[3989, 23929, 27917, 45863, 47857, 63809, 75773, 93719, 95713]
```

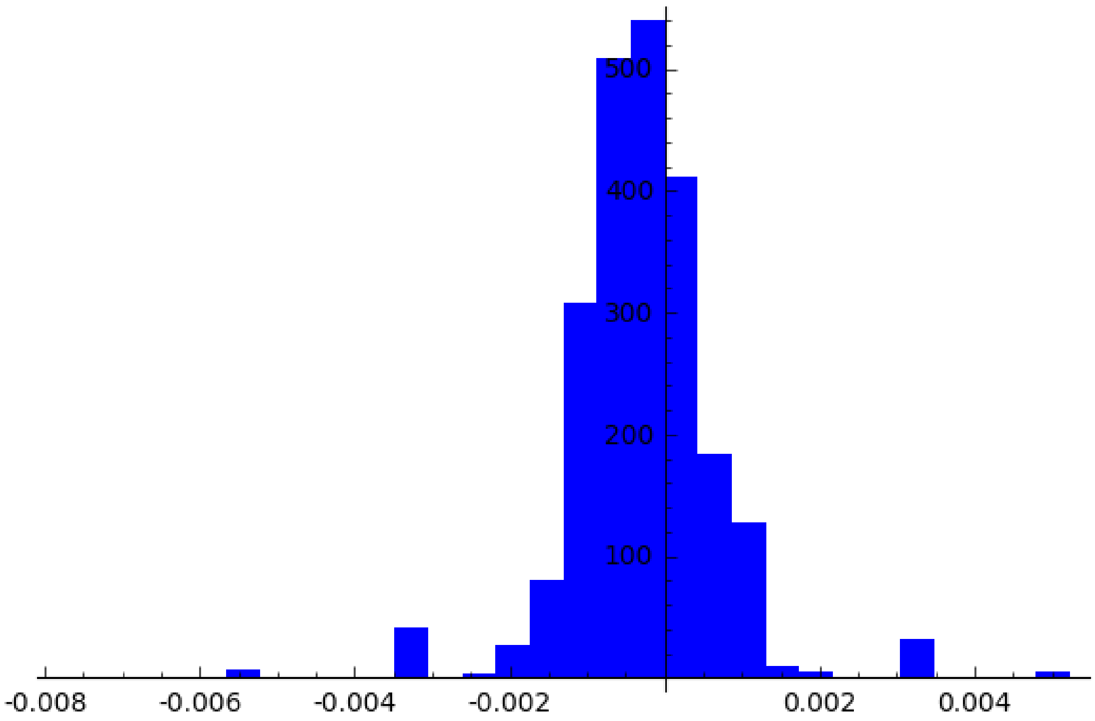
```
In [22]: %%time
         data = []
         for m in ms:
             data += alphas(m, d)

CPU times: user 2.26 s, sys: 8 ms, total: 2.27 s
Wall time: 2.26 s
```

```
In [25]: print len(data)
         t = stats.TimeSeries(data)
         print t.mean()
         t.plot_histogram(bins=30)
```

```
8973
-0.000309958617831
```

```
Out[25]:
```

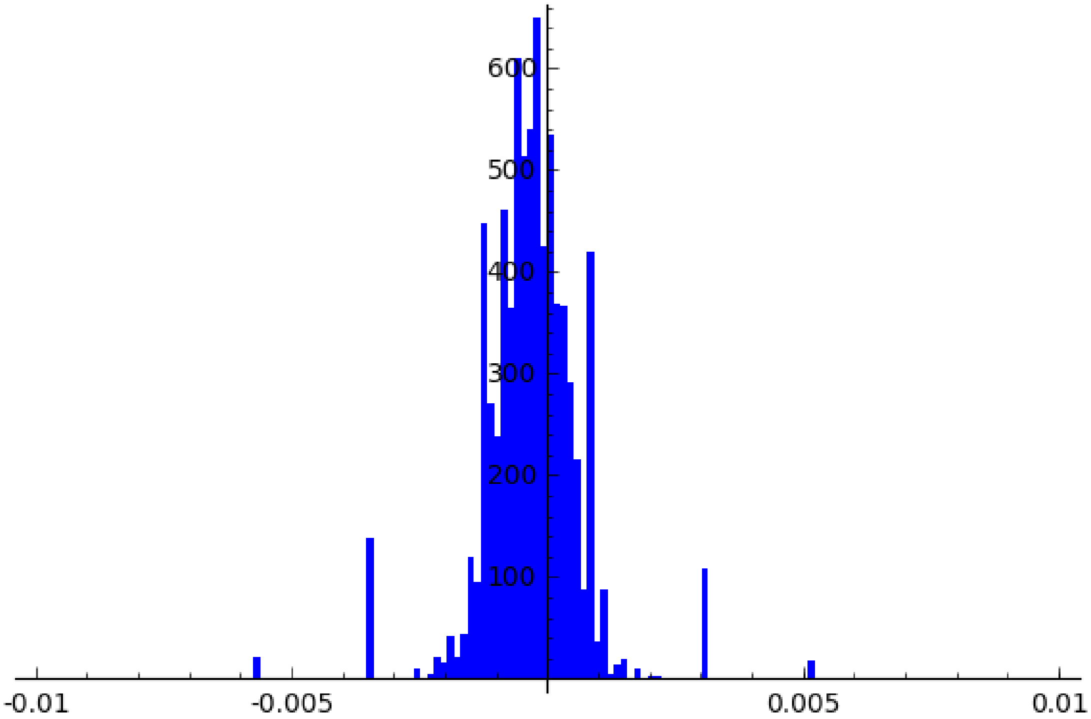


```
In [24]: t.variance()

Out[24]: 9.57983832643914e-07

In [27]: show(t.plot_histogram(bins=100), xmin=-0.01, xmax=0.01)

Out[27]:
```



0.0.5 Now try $d = 2017$

```
In [34]: d = 2017
         # much more ms, since this code is massively faster...
         ms = [m for m in prime_range(3,200000) if gcd(m, 11) == 1 and euler_phi(m) % d == 0]
         print(ms)
```

```
[8069, 36307, 48409, 56477, 72613, 80681, 121021, 129089, 157327, 189599]
```

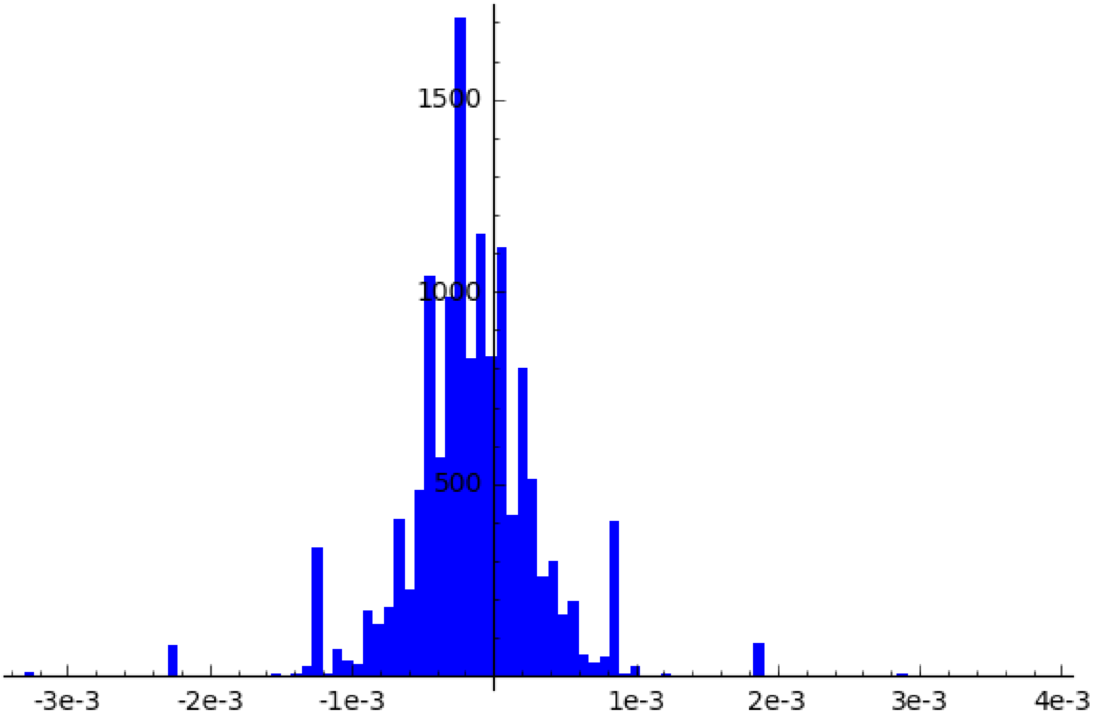
```
In [35]: %%time
         data = []
         for m in ms:
             data += alphas(m, d)
```

```
CPU times: user 5.49 s, sys: 8 ms, total: 5.5 s
Wall time: 5.48 s
```

```
In [36]: print len(data)
         t = stats.TimeSeries(data)
         print t.mean()
         t.plot_histogram(bins=100)
```

```
20170
-0.000148878246339
```

Out[36]:



In [0]: