birkhoff_sum_iet

June 7, 2017

1 Using Sage for computations with interval exchange transformations

Vincent Delecroix, CNRS, LaBRI (Bordeaux, France)

This worksheets is available in notebook and pdf format from http://www.labri.fr/perso/vdelecro/teaching.html

1.1 Short history

A lot of computer experiments have been done since the 90's to understand better interval exchange transformations and translation surfaces: - computation of Rauzy classes (Arnoux, Zorich) - Lyapunov exponents of the Kontsevich-Zorich cocycle (Kontsevich, Zorich) - volumes and Siegel Veech constants (Eskin) - square tiled surfaces (Lelièvre)

(... and some younger experimentators: V. Delecroix, C. Fougeron, P. Hooper)

1.2 Want to learn Sage? Make some computations related to research projects?

Sage days: - (past) Oaxaca Mexico may 2016 - dynamical system Marseille summer school july 2017 (P. Arnoux `pierre@pierrearnoux.fr`) - Grenoble summer school june 2018 (E. Lanneau `erwan.lanneau@univ-grenoble-alpes.fr`) - Toronto end of june 2018 (A. Randecker `anja@math.toronto.edu`) - more?

If interested, get in touch with me (Vincent Delecroix `vincent.delecroix@labri.fr`) and the organizers.

1.3 About this worksheet

The aim of this presentation is to explain how to play with Birkhoff sums over interval exchange transformations using the `surface_dynamics` package. We will mainly illustrate the following theorem of B. Adamczewski

Let $T : [0,1] \to [0,1]$ be a self-similar interval exchange transformation with renormalization matrix M. Let $f : [0,1] \to \mathbb{R}$ be a mean zero function that is constant on each continuity interval of T. Then the Birkhoff sum

$$S_n(f,x) = f(x) + f(Tx) + \ldots + f(T^{n-1}x)$$

is either bounded or

$$\max_{k<n} |S_n(f,x)| \sim C\ n^\theta\ \log(n)^k$$

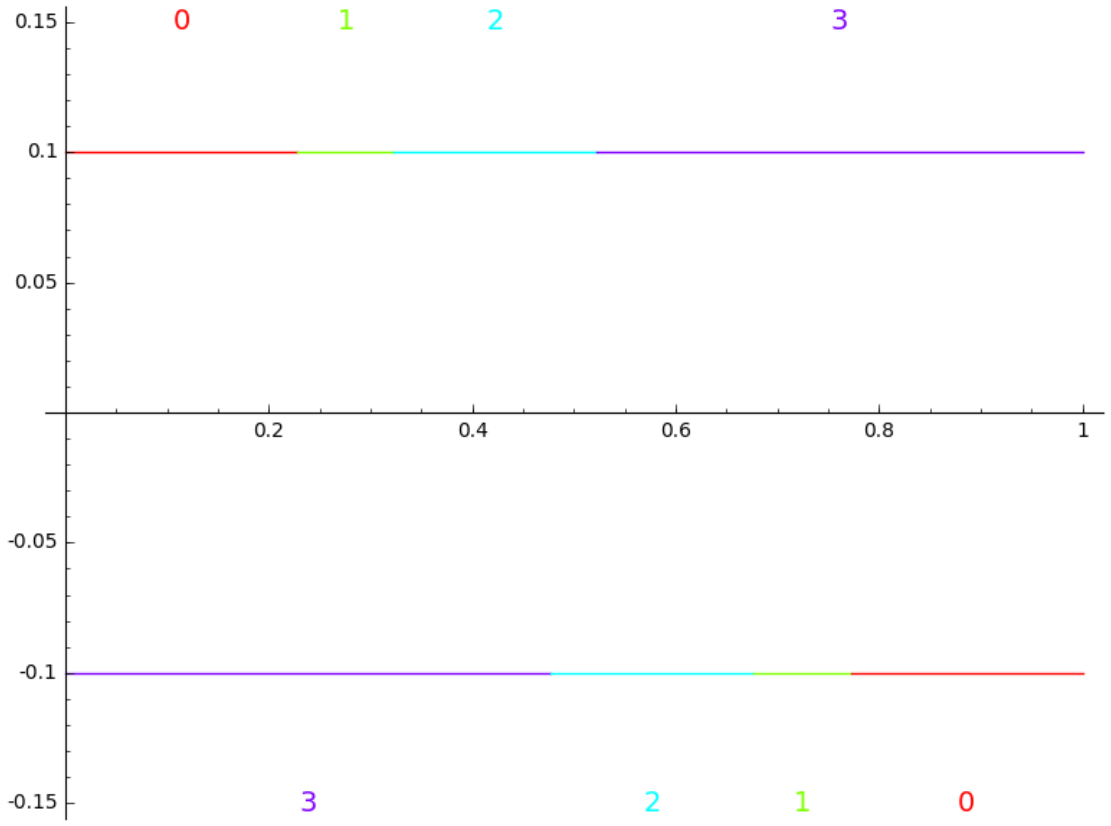
where

- C is some positive constant
- $\theta = \log(|\lambda'|)/\log(|\lambda|)$ is the ratio of logarithms of two eigenvalues of M and
- $k \in \{0, 1, 2, \ldots\}$ is the size of some Jordan block with eigenvalue λ'.

remark: On the one hand, Adamczewski theorem can be considered as a baby case of more general results about Birkhoff sums over generic i.e.t. (Zorich, Forni, Bufetov). But it is important to notice that the conclusion of Adamczewski theorem is much more precise than what is known for generic i.e.t..

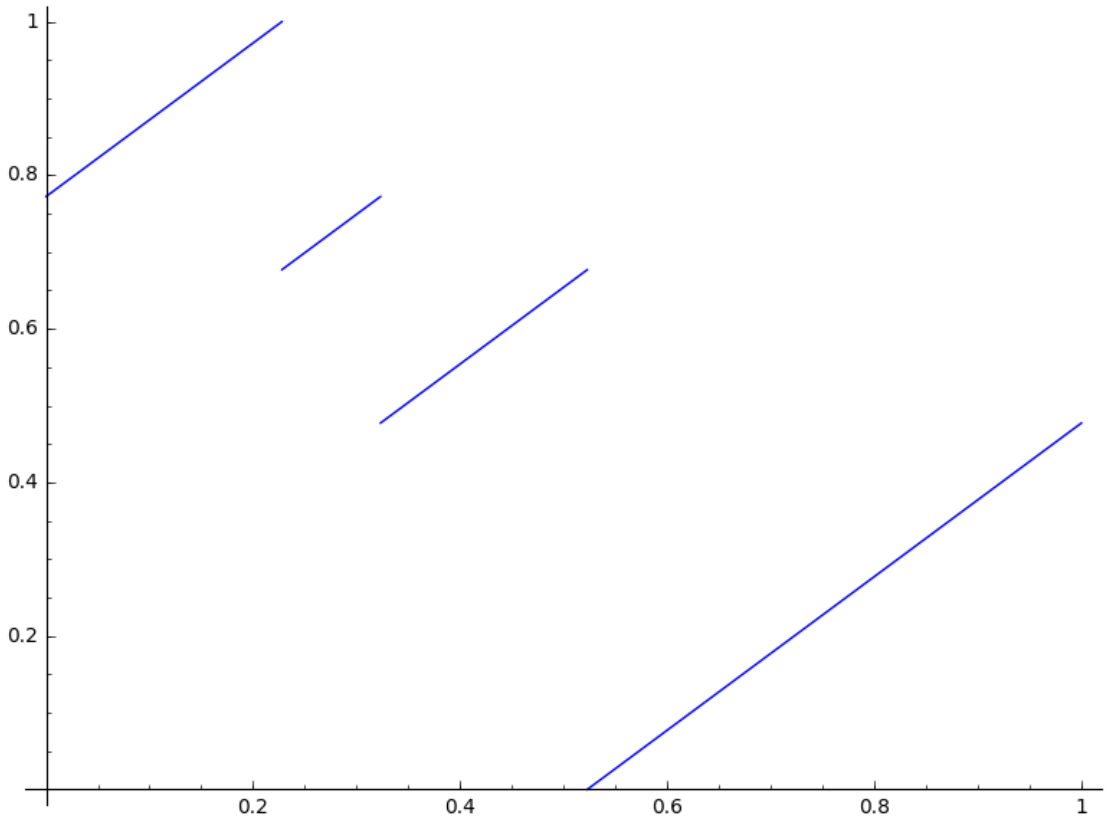
```
In [1]: from surface_dynamics.all import *

In [51]: p = iet.Permutation((0,1,2,3), (3,2,1,0))
         p

Out[51]: 0 1 2 3
         3 2 1 0

In [ ]:

In [338]: # creating a number field
          x = polygen(QQ)                                  # variable of polynomial
          poly = x^4 - 7*x^3 + 13*x^2 - 7*x + 1            # polynomial
          emb = max(poly.roots(AA, False))                 # embedding
          K = NumberField(poly, 'a', embedding=emb)        # number field
          a = K.gen()                                      # generator of number field

In [341]: # a is an exact number!
          a.n(digits=50)

Out[341]: 4.3902568845155136046636000979379800686395248281747

In [344]: # choose some lengths
          lengths = vector((-a^3 + 7*a^2 - 13*a + 7,
                            a^3 - 6*a^2 + 8*a - 4,
                           -2*a^3 + 12*a^2 - 15*a + 4,
                           2*a^3 - 13*a^2 + 20*a - 6))

In [346]: print lengths
          print lengths.n()

(-a^3 + 7*a^2 - 13*a + 7, a^3 - 6*a^2 + 8*a - 4, -2*a^3 + 12*a^2 - 15*a + 4, 2*a^3
(0.227777104234381, 0.0952939852239145, 0.199668914067685, 0.477259996474020)


In [ ]:

In [262]: T = iet.IntervalExchangeTransformation(p, lengths)

In [263]: T.plot_two_intervals()
```

Out[263]:



In [264]: T.plot_function()

Out[264]:

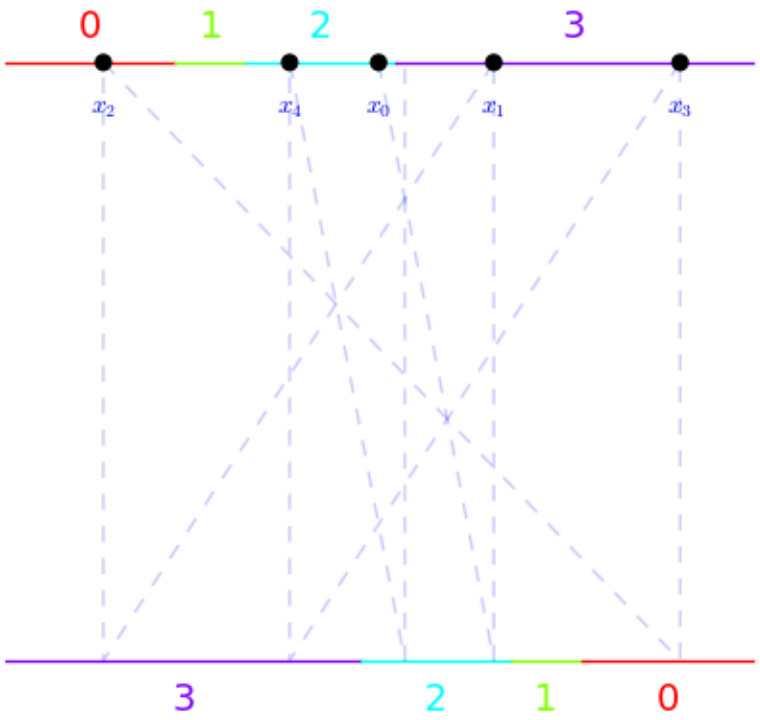


In [ ]:

In [ ]:

In [265]:
```
# plotting an orbit
G = T.plot(interval_height=0.4)
x = 1/2  # initial point
for i in range(5):
    y = T(x)
    G += line2d([(x,0.4),(y,-0.4),(y,0.4)], linestyle='dashed', alpha=0.2
    G += circle((x,0.4), radius=0.01, color='black', fill=True)
    G += text("$x_{%d}$"%i, (x,0.34))
    x = y
```

In [270]:
```
G.show(figsize=(6,4), axes=False)
```




```
In [ ]:
```

```
In [347]: x = 1/2
          coding = []
          for _ in range(10):
              coding.append(T.in_which_interval(x))
              x = T(x)
          print coding
```

```
[2, 3, 0, 3, 2, 3, 0, 3, 1, 3]
```

```
In [ ]:
```

```
In [272]: def birkhoff_sums(T, x, n):
              bsums = [[0],[0],[0],[0]]
              for _ in range(n):
                  i = T.in_which_interval(x)
                  for j in range(4):
```

```
                bsums[j].append(bsums[j][-1] + (i==j))
            x = T(x)
        return bsums
```

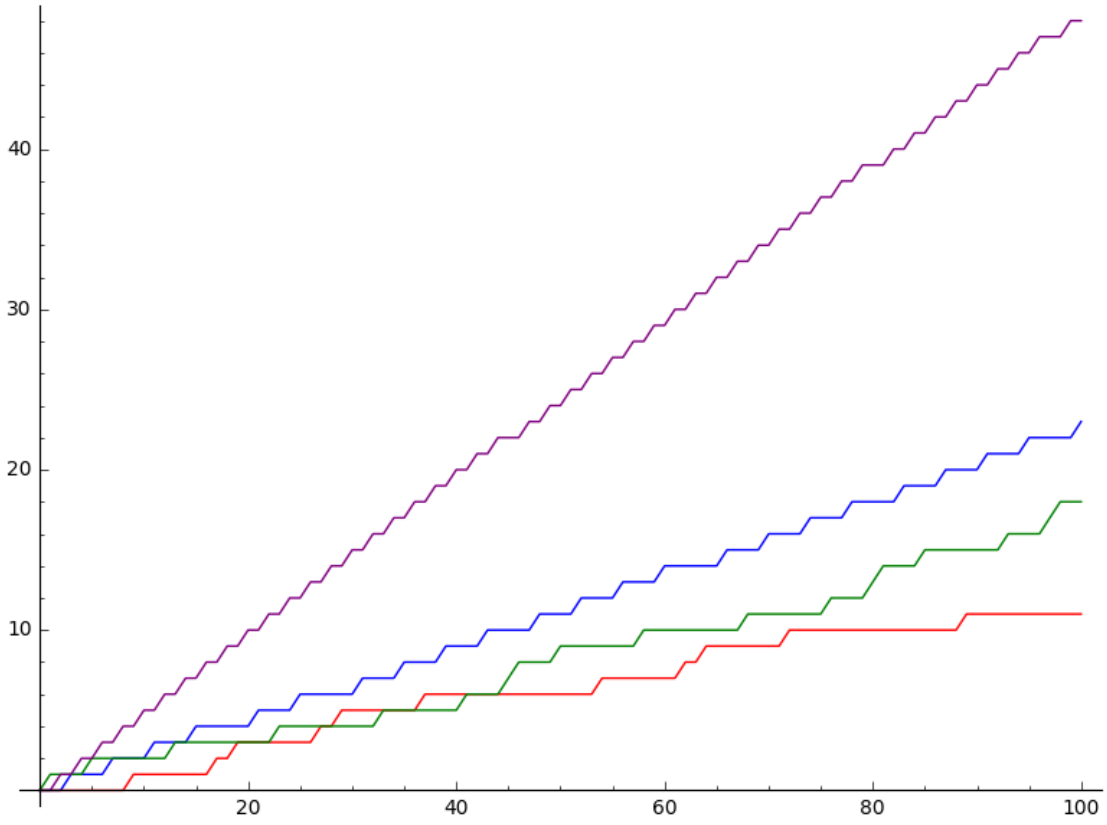
```
In [273]: birkhoff_sums(T, 1/2, 16)
```

```
Out[273]: [[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
           [0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3],
           [0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8]]
```

```
In [274]: b0, b1, b2, b3 = birkhoff_sums(T, 1/2, 100)
          L0 = line2d(list(enumerate(b0)), color='blue')
          L1 = line2d(list(enumerate(b1)), color='red')
          L2 = line2d(list(enumerate(b2)), color='green')
          L3 = line2d(list(enumerate(b3)), color='purple')
```

```
In [275]: (L0 + L1 + L2 + L3)
```

```
Out[275]:
```

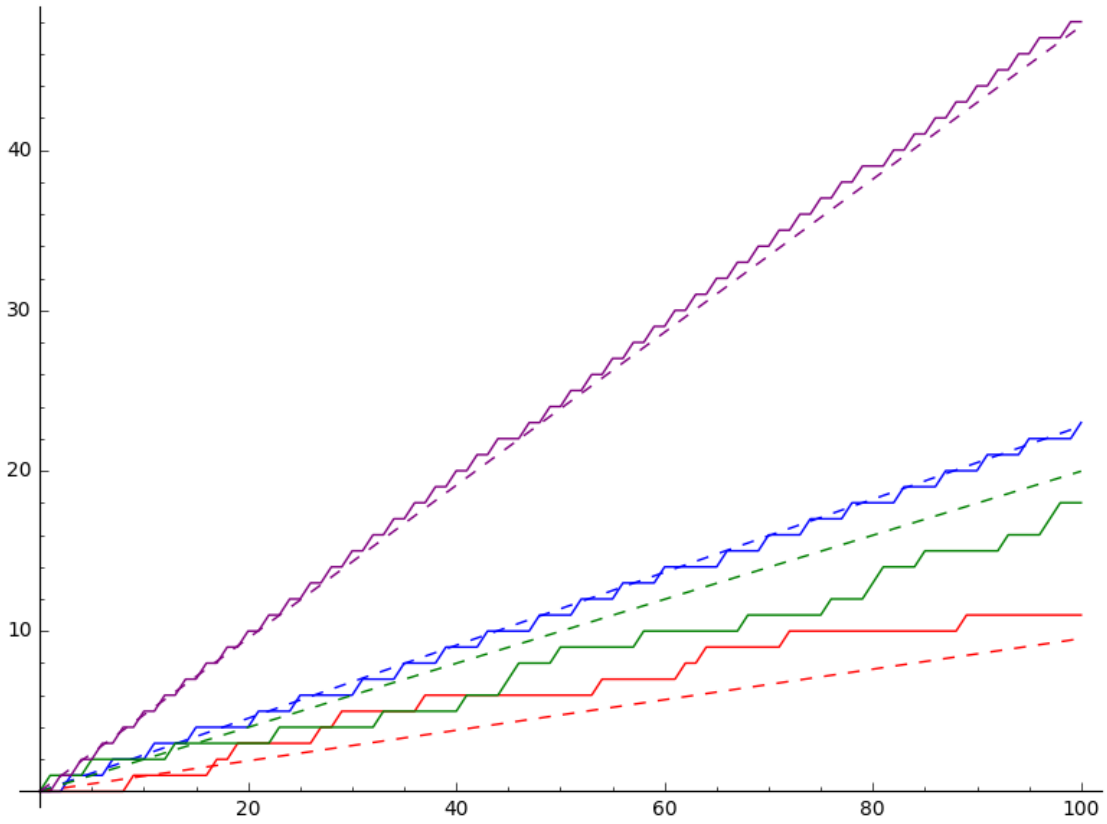



```
In [276]: norm_lengths = vector(lengths) / sum(lengths)
```

In [277]:
```
n = 100
b0, b1, b2, b3 = birkhoff_sums(T, 1/2, n)
L0 = line2d(list(enumerate(b0)), color='blue') + \
     line2d([(0,0),(n, norm_lengths[0]*n)], color='blue', linestyle='dash
L1 = line2d(list(enumerate(b1)), color='red') + \
     line2d([(0,0),(n, norm_lengths[1]*n)], color='red', linestyle='dashe
L2 = line2d(list(enumerate(b2)), color='green') + \
     line2d([(0,0),(n, norm_lengths[2]*n)], color='green', linestyle='das
L3 = line2d(list(enumerate(b3)), color='purple') + \
     line2d([(0,0),(n, norm_lengths[3]*n)], color='purple', linestyle='da
```

In [278]:
```
(L0 + L1 + L2 + L3)
```

Out[278]:



In [ ]:

In [ ]:

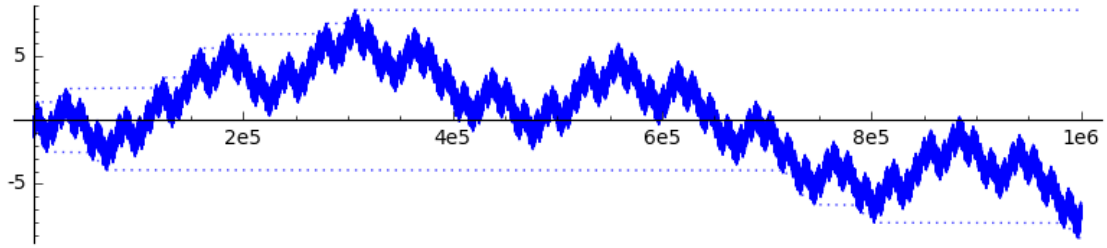
In [279]:
```
def extrema(bsum):
    mins = [(0,0)]
```

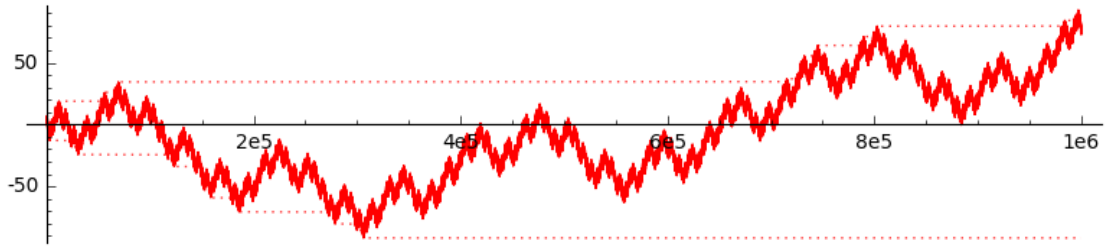
```
            maxs = [(0,0)]
            for i,s in enumerate(bsum):
                if s < mins[-1][1]:
                    mins.append((i,s))
                if s > maxs[-1][1]:
                    maxs.append((i,s))
            mins.append((len(bsum),mins[-1][1]))
            maxs.append((len(bsum),maxs[-1][1]))
            return mins,maxs
```

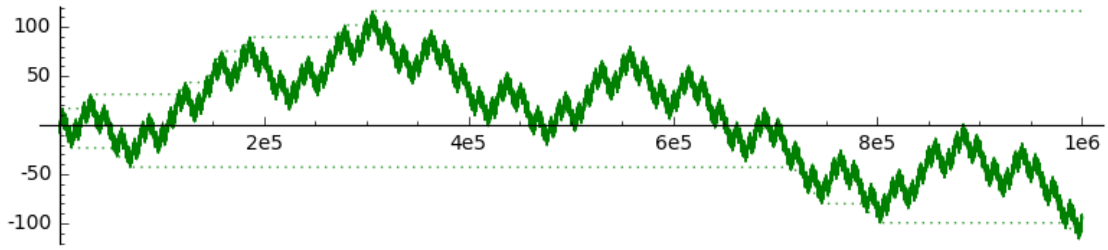
```
In [288]: %%time
          # this takes about a minute
          n = 1000000
          R = RealField(128)
          Tnum = iet.IntervalExchangeTransformation(p, T.lengths().change_ring(R))
          b0, b1, b2, b3 = birkhoff_sums(Tnum, 3/7, n)
          l0, l1, l2, l3 = norm_lengths.n()
          b0 = [x - i * l0 for i,x in enumerate(b0)]
          b1 = [x - i * l1 for i,x in enumerate(b1)]
          b2 = [x - i * l2 for i,x in enumerate(b2)]
          b3 = [x - i * l3 for i,x in enumerate(b3)]
          L0 = line2d(list(enumerate(b0)), color='blue')
          L1 = line2d(list(enumerate(b1)), color='red')
          L2 = line2d(list(enumerate(b2)), color='green')
          L3 = line2d(list(enumerate(b3)), color='purple')

CPU times: user 42.9 s, sys: 409 ms, total: 43.3 s
Wall time: 42.9 s
```

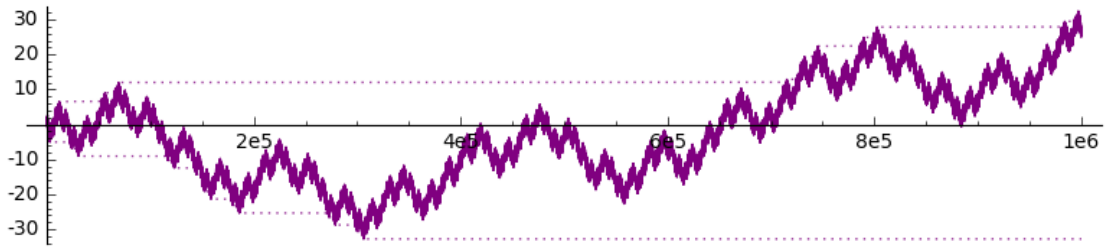
```
In [289]: opts = dict(linestyle='dotted', alpha=0.8)
          m0, M0 = extrema(b0)
          m1, M1 = extrema(b1)
          m2, M2 = extrema(b2)
          m3, M3 = extrema(b3)
          E0 = line2d(m0, color='blue', **opts) + line2d(M0, color='blue', **opts)
          E1 = line2d(m1, color='red', **opts) + line2d(M1, color='red', **opts)
          E2 = line2d(m2, color='green', **opts) + line2d(M2, color='green', **opts
          E3 = line2d(m3, color='purple', **opts) + line2d(M3, color='purple', **op
```

```
In [290]: (L0 + E0).show(figsize=(8,2))
          (L1 + E1).show(figsize=(8,2))
          (L2 + E2).show(figsize=(8,2))
          (L3 + E3).show(figsize=(8,2))
```


5
-5
2e5
4e5
6e5
8e5
1e6


50
-50
2e5
4e5
6e5
8e5
1e6

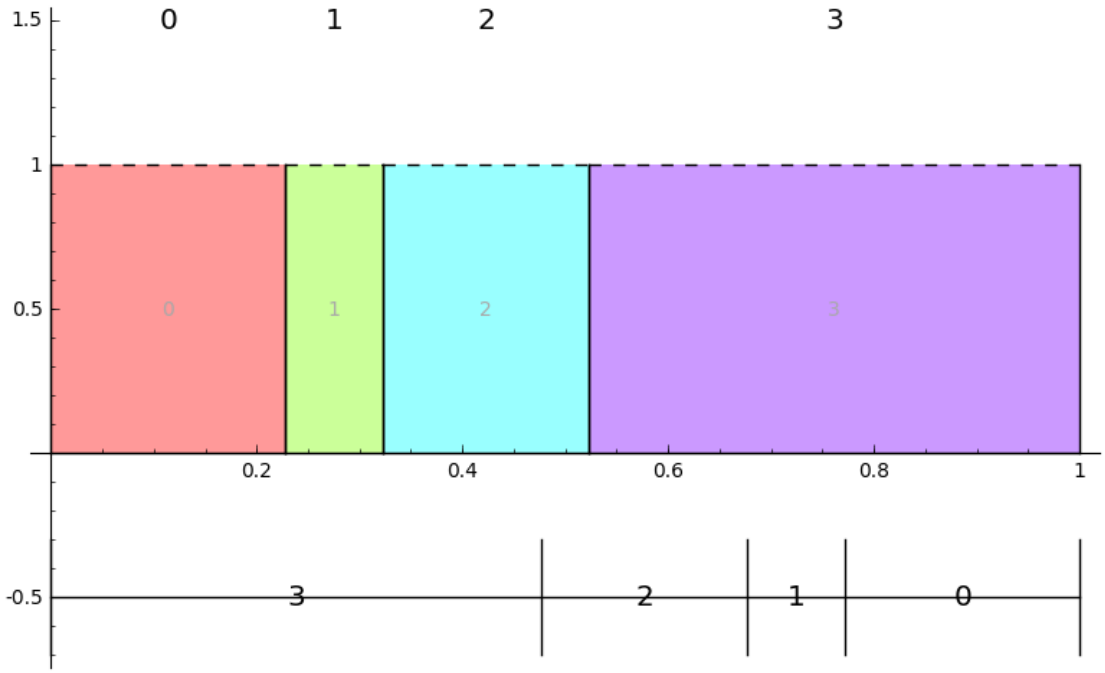

100
50
-50
-100
2e5
4e5
6e5
8e5
1e6

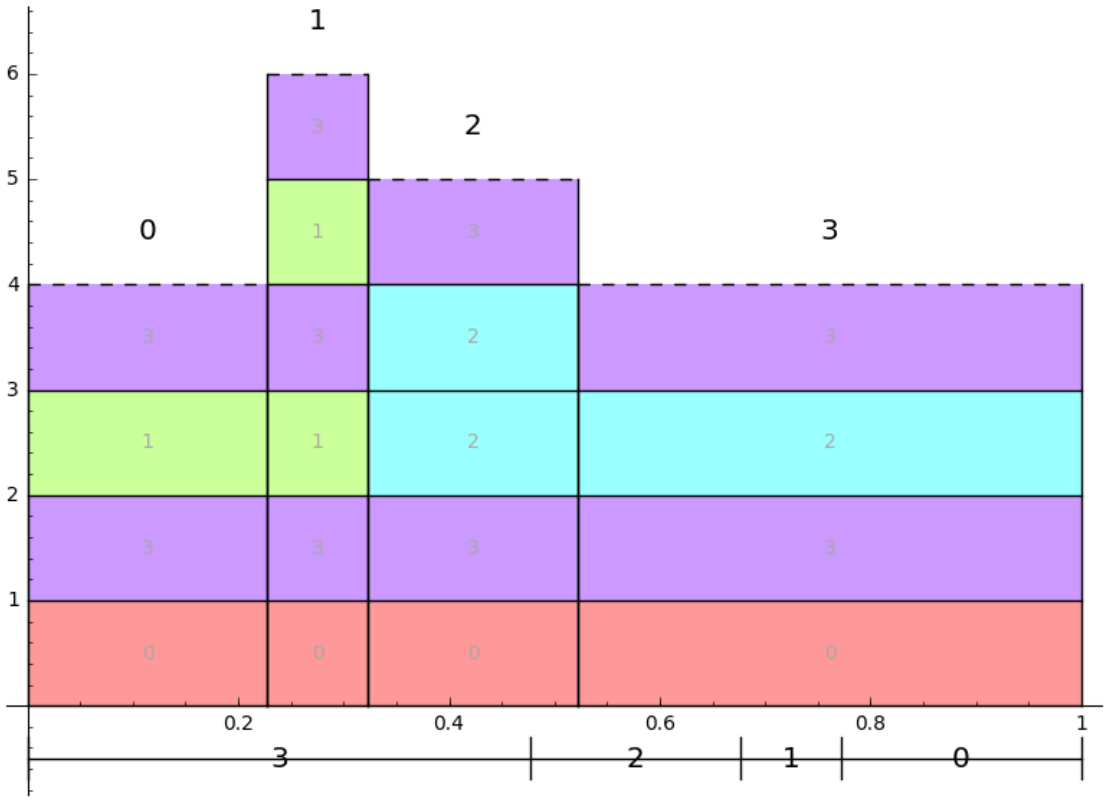

30
20
10
-10
-20
-30
2e5
4e5
6e5
8e5
1e6

In [ ]:

In [ ]:

In [330]:
```
# the iet we started with is a self similar iet
TT = iet.IntervalExchangeTransformation(p, vector(lengths) / lengths[0])
T.plot_towers(iterations=0).show(aspect_ratio=0.28)
TT.plot_towers(iterations=8).show(aspect_ratio=0.1)
```






```
In [334]: # the iet was actually constructed from a loop in
          # the Rauzy diagram (Rauzy, Veech)
          path = 'ttbtbbtb'
          R = p.rauzy_diagram()
          g = R.path(p, *path)
          eigs = sorted(g.matrix().eigenvalues())
          eigs
          g.matrix()

Out[334]: [1 1 1 1]
          [1 2 0 0]
          [0 0 2 1]
          [2 3 2 2]

In [337]: # By Adamczewski this should be the exponent of the polynomial
          # growth rate of trajectories
          log(eigs[-2].n()) / log(eigs[-1].n())

Out[337]: 0.411385002063387

In [336]: print [log(abs(float(y)))/log(float(x)) for x,y in m1[-5:]]
          print [log(abs(float(y)))/log(float(x)) for x,y in M1[-5:]]
```

```
print [log(abs(float(y)))/log(float(x)) for x,y in m2[-5:]]
print [log(abs(float(y)))/log(float(x)) for x,y in M2[-5:]]
print [log(abs(float(y)))/log(float(x)) for x,y in m3[-5:]]
print [log(abs(float(y)))/log(float(x)) for x,y in M3[-5:]]
```

```
[0.35740080964063825, 0.3574832506498919, 0.3575656056590716, 0.3576478748471031, 0
[0.327059355553862, 0.3272467348559632, 0.32758194506161326, 0.32776797413339925, 0
[0.34382373312399106, 0.3439476174660387, 0.3440724487609351, 0.3441970650826055, 0
[0.3758928077121051, 0.3760304170592185, 0.37630555875050753, 0.3764424472099946, 0
[0.27534251950091365, 0.275617968732891, 0.2761086502624149, 0.27638141305292535, 0
[0.2516033718369013, 0.25170517989437097, 0.251806844855031, 0.2519083671200714, 0.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: