Luminy2026FlatsurfPresentation

November 11, 2026

1 SageMath, flatsurf and admcycles

Luminy demo, nov. 2026

- SageMath is (an open source) math software
- flatsurf is a suite of libraries for SageMath for computing with cone flat surfaces (in particular translation surfaces), cone hyperbolic surfaces, interval exchange transformations, linear subvarietes (or AIS in Paul's talk), …
- admcycles is a library for SageMath for algebraic computation with the cohomology of moduli spaces (and more specifically the tautological subrings)

1.1 SageMath

- *huge* software which gathers a lot of other math softwares (flint, PARI/GP, mpfr, GAP, …)
- interface in *Python*
- mostly exact computations (number theory, graphs, combinatorics, …)

[1]:
```
1 + 1
```

[1]:
```
2
```

[2]:
```
K = NumberField(x^2 - 5, 'sqrt5', embedding=AA(5).sqrt())
sqrt5 = K.gen()
sqrt5 - 1
```

[2]:
```
sqrt5 - 1
```

[3]:
```
almost_one = 21115139763499233108905110266449 * sqrt5 -␣
↪472148878655931179109174234129 97
almost_one.numerical_approx()
```

[3]:
```
1.00000000000000
```

[4]:
```
almost_one.numerical_approx() == 1
```

[4]:
```
True
```

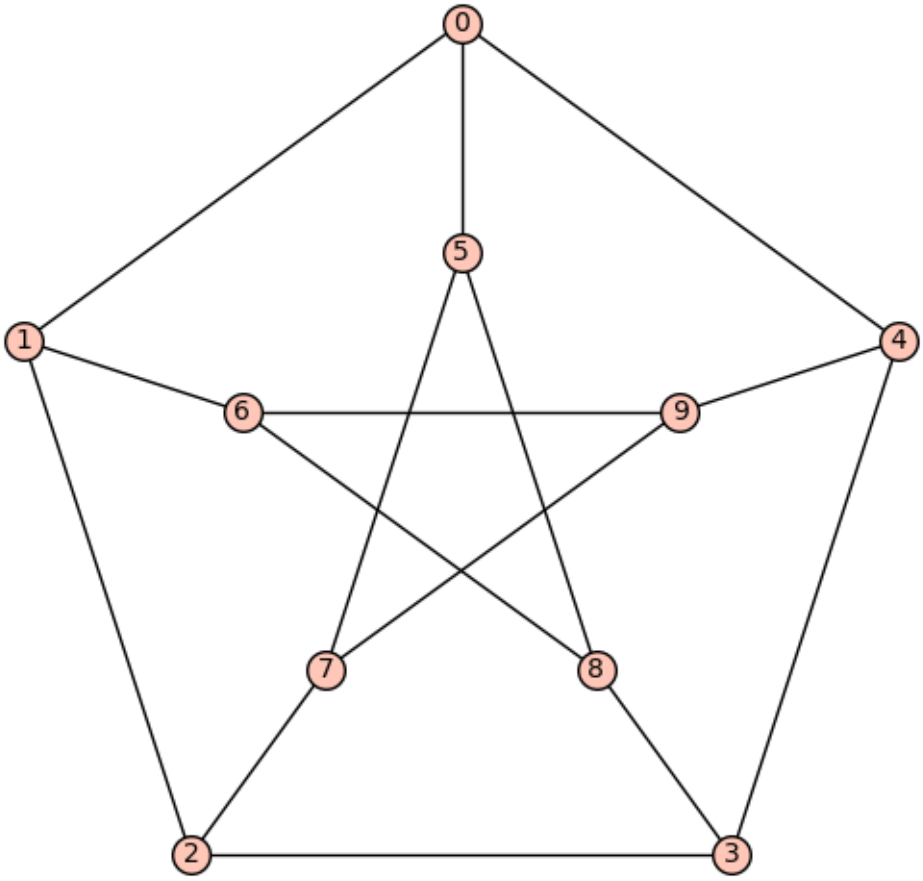
[5]:
```
almost_one == 1
```

[5]:
```
False
```

[ ]:

[6]:
```
G = graphs.PetersenGraph()
G.plot()
```

[6]:



[ ]:

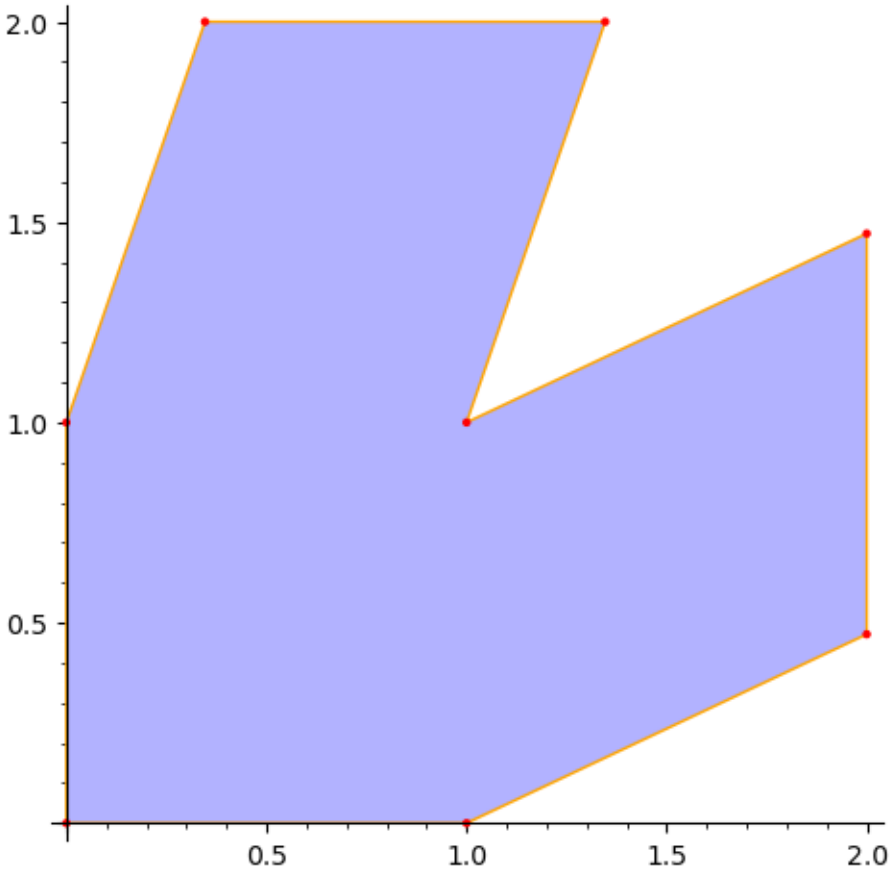
1.2 Delaunay triangulations (`flatsurf`)

[7]:
```
import flatsurf
```

[8]:
```
s = AA(2).sqrt() / 3
t = AA(3).sqrt() / 5
K, (s, t), _ = number_field_elements_from_algebraics([s, t], minimal=True,␣
 ↪embedded=True)
P = flatsurf.
 ↪Polygon(vertices=[(0,0),(1,0),(2,s),(2,1+s),(1,1),(1+t,2),(t,2),(0,1)])
```

```
P.plot()
```

[8]:




```
S = flatsurf.MutableOrientedSimilaritySurface(K)
S.add_polygon(P)
S.glue((0,0),(0,5))
S.glue((0,1),(0,3))
S.glue((0,2),(0,7))
S.glue((0,4),(0,6))
S.set_immutable()
```

[9]:

```
S
```

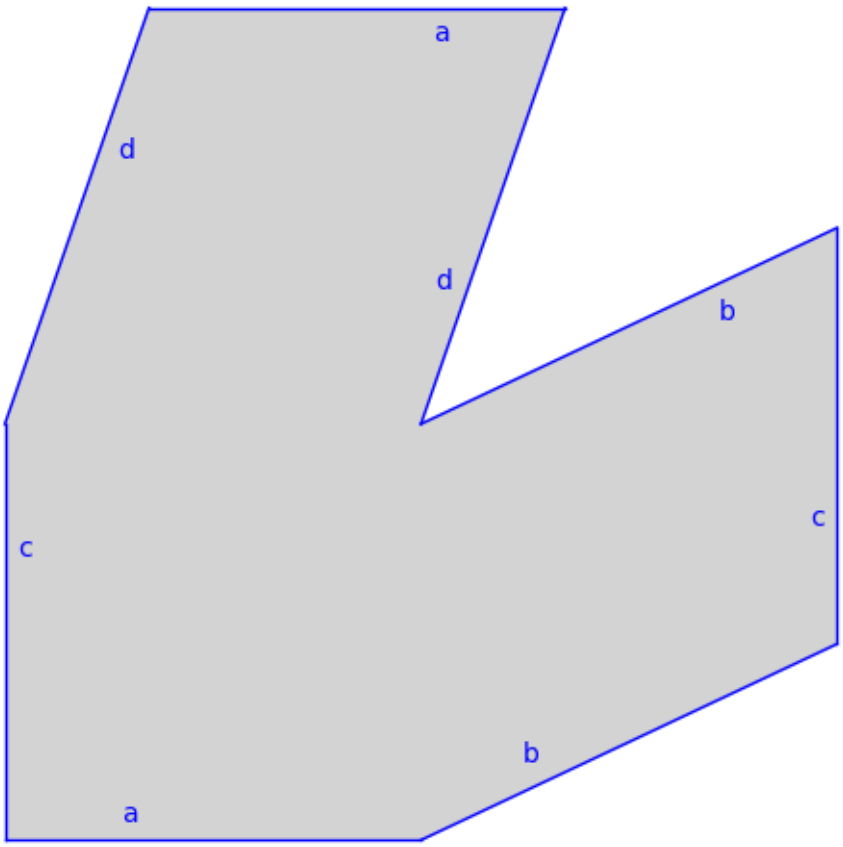
[10]:

[10]: Translation Surface in H_2(2) built from a non-convex octagon

```
S.plot(edge_labels="letter", polygon_labels=False)
```

[11]:

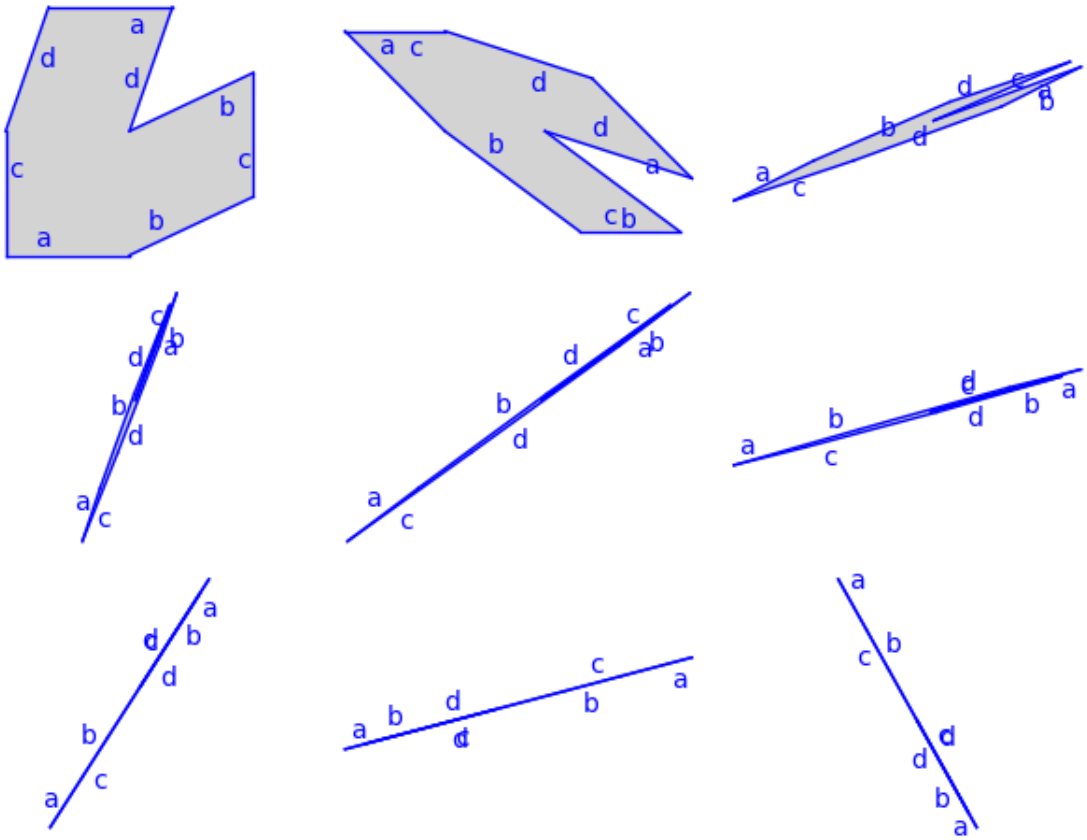
[11]:



[12]:
```
# NOTE: make a random walk by applying matrices in SL(2,Z)
# we use the standard parabolic generators and use a random
# product of three of them in the steps
steps = [matrix(ZZ, 2, [1,1,0,1]),
         matrix(ZZ, 2, [1,-1,0,1]),
         matrix(ZZ, 2, [1,0,1,1]),
         matrix(ZZ, 2, [1,0,1,-1])]
walk = [S]
SS = S
for _ in range(8):
    SS = choice(steps) * choice(steps) * choice(steps) * SS
    walk.append(SS)
```

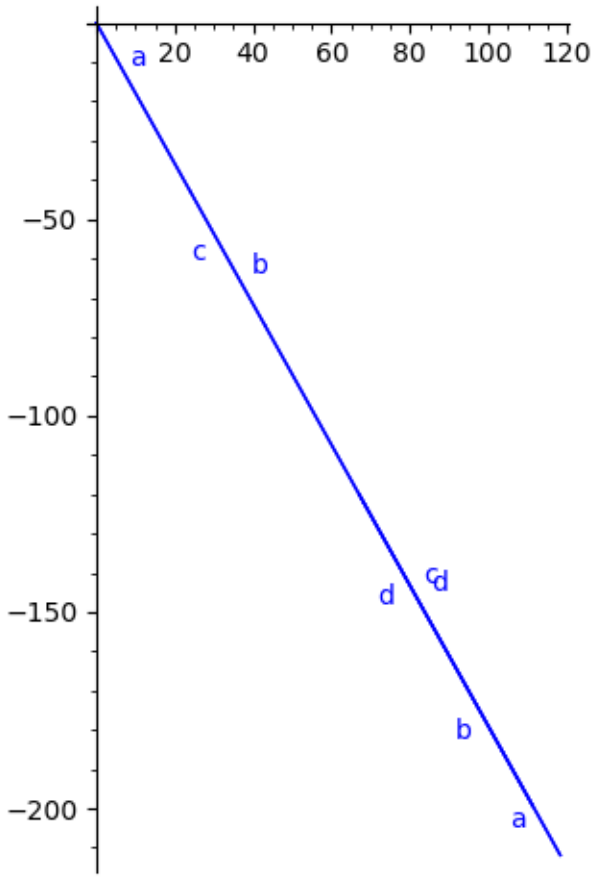
[13]:
```
graphics_array([SS.plot(edge_labels="letter", polygon_labels=False) for SS in␣
↪walk], 3, 3)
```

[13]:

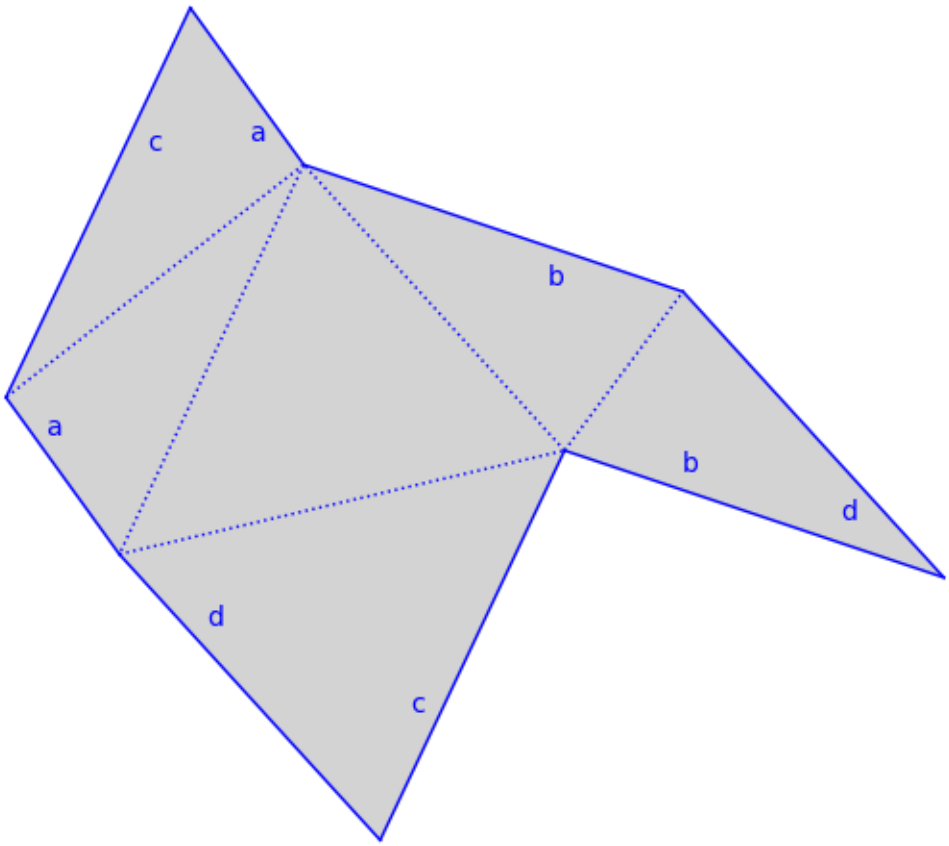


[14]:
```
# NOTE: there is a lot of lazyness in the Delaunay code computation. To␣
 ↪actually trigger
# the computation, one must iterate over the polygons
S2 = walk[-1]
D = S2.delaunay_decompose().codomain()
polygons = list(D.polygons())
```

[15]:
```
S2.plot(edge_labels="letter", polygon_labels=False).show(axes=True)
```



[16]:
```
D.plot(edge_labels="letter", polygon_labels=False)
```
[16]:



[ ]:

1.3 Saddle connections (`flatsurf`)

[17]:
```
import flatsurf
s = AA(2).sqrt() / 3
t = AA(3).sqrt() / 5
K, (s, t), _ = number_field_elements_from_algebraics([s, t], minimal=True,␣
 ↪embedded=True)
P = flatsurf.
 ↪Polygon(vertices=[(0,0),(1,0),(2,s),(2,1+s),(1,1),(1+t,2),(t,2),(0,1)])
S = flatsurf.MutableOrientedSimilaritySurface(K)
S.add_polygon(P)
S.glue((0,0),(0,5))
S.glue((0,1),(0,3))
S.glue((0,2),(0,7))
S.glue((0,4),(0,6))
S.set_immutable()
```

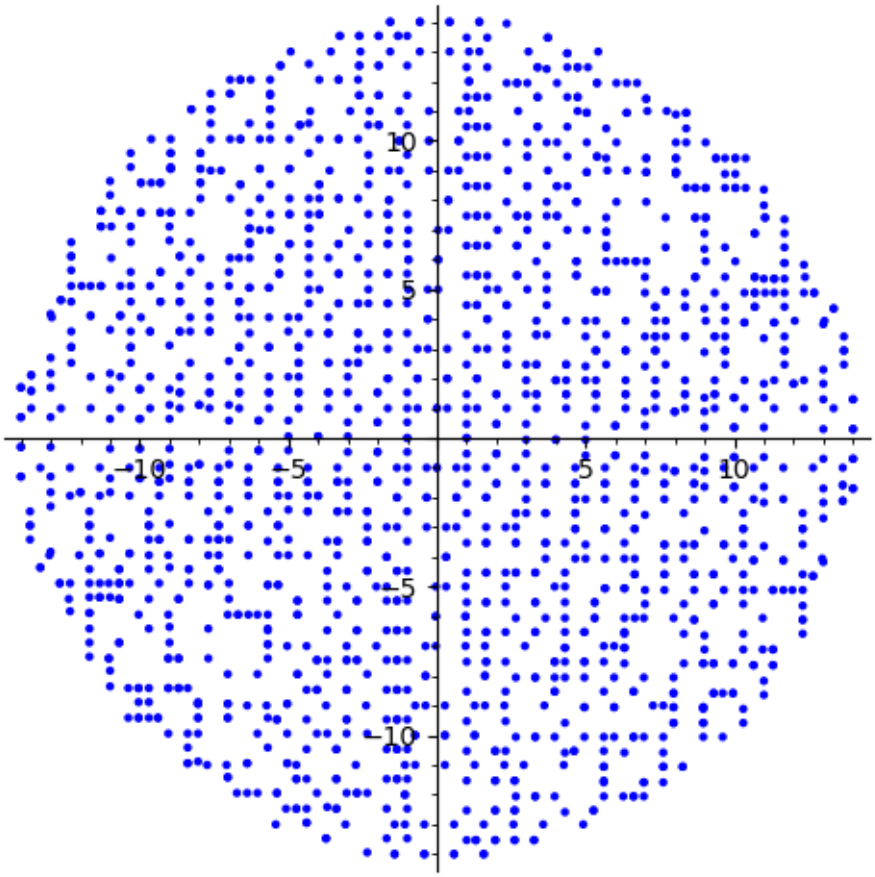
```
S
```

[17]:
```
Translation Surface in H_2(2) built from a non-convex octagon
```

[18]:
```
St = S.triangulate().codomain().relabel()
```

[19]:
```
scs = [sc for sc in St.saddle_connections(200)]
```

[20]:
```
point2d([sc.holonomy() for sc in scs]).show(aspect_ratio=True)
```



[ ]:
```

```

1.4 Interval exchange transformations, Lyapunov exponents and deviation of Birkhoff sums (Kontsevich-Zorich)

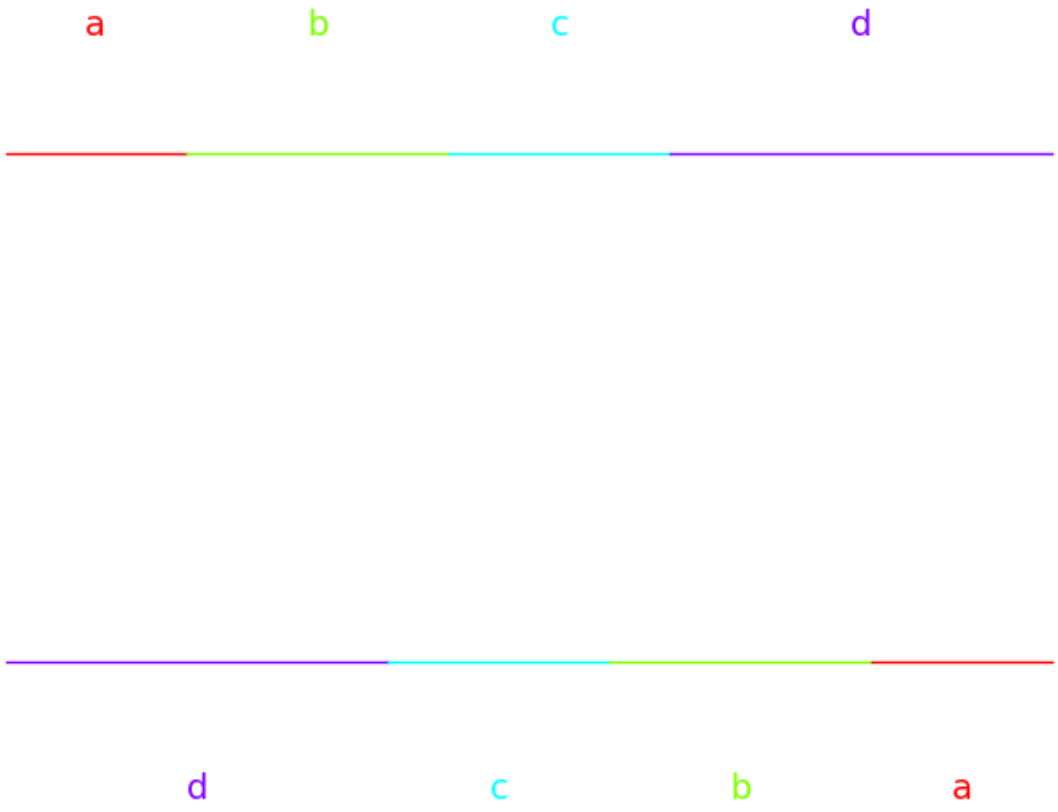
[21]:
```
import surface_dynamics
```

[22]:
```
s = AA(2).sqrt() / 3
t = AA(3).sqrt() / 5
```

```
K, (s, t), _ = number_field_elements_from_algebraics([s, t], minimal=True,␣
↪embedded=True)
```

[23]:
```
p = surface_dynamics.iet.Permutation(['a','b','c','d'],['d','c','b','a'])
T = surface_dynamics.iet.IntervalExchangeTransformation(p, [13/5, 10/3 + s, 5/2␣
↪+ 2*t, 33/7 + s + t])
```

[24]:
```
T.plot().show(axes=False)
```

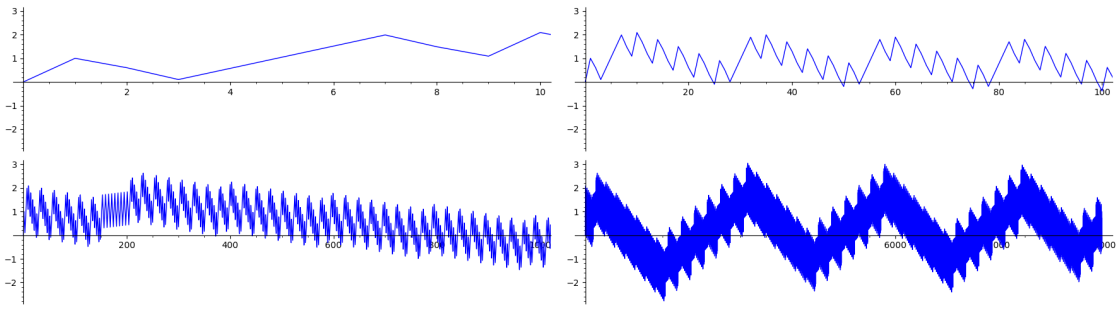


[25]:
```
# basis of piecewise-constant functions with zero mean
zero_mean_basis = matrix(T.base_ring(), 1, 4, T.lengths()).
↪right_kernel_matrix().rows()
zero_mean_basis
```

[25]:
```
[(1, 0, 0, -316136730/7273695263*a^3 + 176398677/7273695263*a^2 +
1996144605/14547390526*a - 8855991963/14547390526),
 (0, 1, 0, 279546155/14547390526*a^3 + 439570425/14547390526*a^2 -
801760575/14547390526*a - 11178980575/14547390526),
 (0, 0, 1, -247700880/7273695263*a^3 - 914679045/14547390526*a^2 +
2804888625/29094781052*a - 11689465599/29094781052)]
```

[26]:
```
# compute Birkhoff sum along the orbit of x0
# a vector encoding a piecewise-constant function must be converted to
# a dictionary letter -> entry  as in_which_interval() returns letters.
f = zero_mean_basis[0] - 1/2 * zero_mean_basis[1] + s * zero_mean_basis[2]
f = dict(zip(p.alphabet(), f))
x0 = 1/3 + s/7
x = x0
B = [0]
for _ in range(10000):
    B.append(B[-1] + f[T.in_which_interval(x)])
    x = T(x)
```

[27]:
```
graphics = []
for xmax in [10, 100, 1000, 10000]:
    L = list_plot(B, plotjoined=True)
    L.axes_range(None, xmax, None, None)
    graphics.append(L)
graphics_array(graphics, 2, 2).show(figsize=(18,5))
```



[ ]:
```
```

[28]:
```
p.lyapunov_exponents_approx(nb_experiments=100, nb_iterations=2**20)
```

[28]:
```
[1.0002794309955632, 0.3332018869671057]
```

[ ]:
```
```

2 admcycles

- `admcycles`: computes in the cohomology of $\mathcal{M}_{g,n}$ (actually its Deligne-Mumford compactification)
- `diffstrata` (inside `admcycles`): is about the cohomology of strata of translation surfaces $\mathcal{H}(\kappa)$ (actually its BCGGM compactification)

`admcycles` works in a subring of the even cohomology called the *tautological ring*. Its implementation is based on the work of A. Pixton who found a combinatorial algebra called the *strata algebra* such that the tautological ring is its quotient. Understanding the exact set of relations is a big open question and there was a long standing conjecture of A. Pixton describing them which turns out to not be true in genus `g=25`. The set of relations used to work with the tautological ring in admcycles is still the conjectural one so that it gives *wrong answers* (ie an answer `False` to `A = B` might actually be `True` because of forgotten relations). Though, genus 25 is out of reach of the library and in the range of `(g,n)` for which computations can be performed the conjectural relations give actually all relations.

[29]:
```
import admcycles
import admcycles.diffstrata
```

[ ]:
```
```

[30]:
```
# subring of the even cohomology ring
R = admcycles.TautologicalRing(2, 1)
```

[31]:
```
R.psi(1)
```

[31]:
```
Graph :      [2] [[1]] []
Polynomial : psi_1
```

[32]:
```
# getting generators in a given degree
g0, g1, g2, g3 = R.generators(1)
```

[33]:
```
print(g0)
```

```
Graph :      [2] [[1]] []
Polynomial : (kappa_1)_0
```

[34]:
```
print(g1)
```

```
Graph :      [2] [[1]] []
Polynomial : psi_1
```

[35]:
```
print(g2)
```

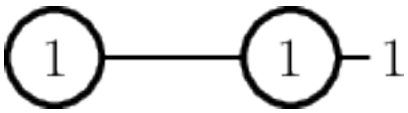
```
Graph :      [1, 1] [[3], [1, 4]] [(3, 4)]
Polynomial : 1
```

[36]:
```
print(g3)
```

```
Graph :      [1] [[3, 4, 1]] [(3, 4)]
Polynomial : 1
```

[37]:
```
# the stable graph for generator g2
sg2 = admcycles.StableGraph([1,1], [[3], [1,4]], [(3,4)])
sg2.tikz(standalone=True)
```

[37]:



[38]:
```
# the stable graph for generator g3
sg3 = admcycles.StableGraph([1], [[3,4,1]], [(3,4)])
sg3.tikz(standalone=True)
```

[38]:


[39]:
```
# the multiplication in the tautological ring is one of the involved algorithm
(g0 + 3*g1) * (1 + g2) * (1 + g3)
```

[39]:
```
Graph :      [2] [[1]] []
Polynomial : (kappa_1)_0 + 3*psi_1

Graph :      [1] [[3, 4, 1]] [(3, 4)]
Polynomial : (kappa_1)_0 + 3*psi_1

Graph :      [1, 1] [[3], [1, 4]] [(3, 4)]
Polynomial : (kappa_1)_0 + (kappa_1)_1 + 3*psi_1

Graph :      [0, 1] [[3, 5, 6], [1, 4]] [(3, 4), (5, 6)]
Polynomial : (kappa_1)_1 + 3*psi_1

Graph :      [1, 0] [[3], [1, 4, 5, 6]] [(3, 4), (5, 6)]
Polynomial : (kappa_1)_0 + (kappa_1)_1 + 3*psi_1
```

[ ]:

[40]:
```
S = admcycles.diffstrata.Stratum([2],1)
```

[41]:
```
S.masur_veech_volume()
```

[41]:
```
1/120*pi^4
```

[42]:
```
# exercise: formula for Siegel-Veech
```

[ ]: