Approximation Diophantienne et exposant de Lyapounov

Soit $\omega = (\omega_1, \omega_2, \omega_3) \in \mathbb{R}_3^+$ t.q $\|\omega\|_\infty = 1$ (i.e. $\omega_3 = 1$).
$= (\omega_1, \omega_2, 1)$

$\boxed{d=2}$ pour simplifier ($\overset{*}{\text{ou}}$ t.q $\|\omega\|_1 = 1$, i.e. $\omega_3 = 1 - \omega_1 - \omega_2$)

But: Construire une suite $(p_1(n), p_2(n), q(n)) = u(n) \in \mathbb{N}^3$
(classique) t.q. $x(n) = \frac{u(n)}{\|u(n)\|_\infty} = \left(\frac{p_1(n)}{q(n)}, \frac{p_2(n)}{q(n)}, 1\right) \to (\omega_1, \omega_2, 1)$

But*(new) et telle que $u(n+1) - u(n) \in \{(1,0,0), (0,1,0), (0,0,1)\}$
$\hat{m}$ chose avec norme 1 et $\ast$

Convergence faible $\lim_{n\to\infty} \|\omega - x(n)\| = 0$

Convergence forte $\lim_{n\to\infty} \| q(n)\cdot\omega - \underbrace{(p_1(n), p_2(n), q(n))}_{u(n)} \| = 0$

Convergence forte* (discrépance bornée) $(\exists C)$ $\| \|u(n)\|_1 \cdot \omega - u(n) \| < C$

Dirichlet Theorem $\forall (w_1, w_2, \ldots w_d) \in [0,1]^d \setminus \mathbb{Q}^d$ has infinitely many approximations of the form $\left(\frac{p_1}{q}, \ldots, \frac{p_d}{q}\right)$ s.t. for all $1 \le j \le d$ $\left|w_j - \frac{p_j}{q}\right| \le \frac{1}{q^{1+1/d}}$

$\Rightarrow$ Existence of strongly convergent sequences

Rem $d=1$, fractions continues, résolu
$d > 1$, + difficile, $\exists$ (almost everywhere strongly conv. results)

Pour évaluer une approximation x:
(Roth exponent) $\eta(x, \omega) = -\frac{\log\|\omega - x\|_\infty}{\log q}$ $\left(\Leftrightarrow \|\omega - x\| = q^{-\eta(x,\omega)}\right)$

Best approximation exponent for ω using algo A (Lagarias)

$$\eta_A(\omega) := \limsup_{n\to\infty} \left\{ \max_{1\le i\le d+1} \eta(x_i(n), \omega) \right\}$$

Uniform approximation exponent for ω using algo A

$$\eta_A^{*}(\omega) := \liminf_{n\to\infty} \left\{ \min_{1\le i\le d+1} \eta(x_i(n), \omega) \right\}$$

H1 : Ergodicity H2 : Covering Property H3 : Semi weak Convergence H4 : Boundedness H5 : Partial quotient mixing

Theorem (Lagarias, 1993) Let A be a MCF algorithm on $[0,1]^d$ that satisfies (H1 – H5) hypothesis, then

$$\eta_A(\omega) \geq \eta^*_A(\omega) = 1 - \frac{\theta_2}{\theta_1} \quad \text{a.e.}$$

where $\theta_1 > \theta_2$ are the two largest Lyapunov exponents.

Note $\quad 1 \leq \eta^*_A(\omega) \leq 1 + \frac{1}{d}$ for any algo

An algo which realises $\eta^*_A = 1 + \frac{1}{d}$ is an optimal algorithm

∴ CF optimal, $d \geq 2$: open problem

Multi Theorem

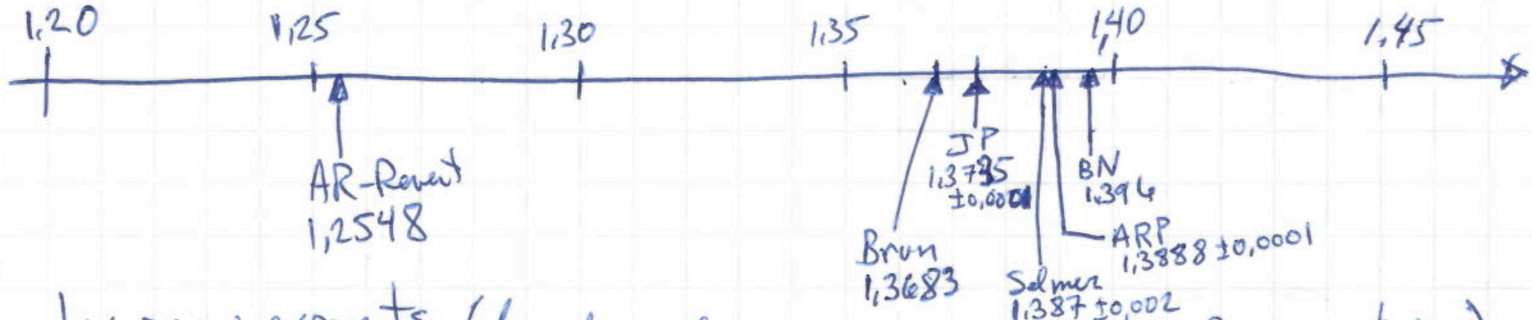
(Lagarias 93, A = Jacobi Perron, Selmer)
(Baladi, Nogueira 96, A = Selmer multiplicatif qui aime $\leq \frac{1}{2}$)
(Bruin, Fokkink, Kraaikamp 2013 (preprint)) A = Gen. Selmer

Hyp (H1 – H5) are satisfied!

Calculs

$$\eta^*_A = 1 - \frac{\theta_2}{\theta_1} = \begin{cases} 1{,}374 \pm 0{,}002 & \text{pour JP} \\ 1{,}387 \pm 0{,}002 & \text{pour Selmer} \\ 1{,}396 & \text{pour l'algo de Baladi Nogueira 96} \end{cases}$$

(Baldwin '92)



Lyapunov exponents (log des valeurs propres de produit infini de matrices)

... Formulas from (Delecroix, Berthé, 2013)

(Assuming ergodicity, log-integrability)

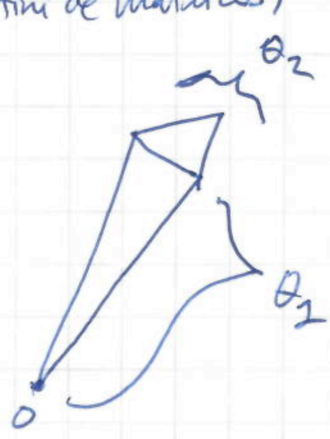
$$\theta_1 = \lim_{n \to \infty} \frac{\log \|M_n(\omega)\|}{n}$$

(Assuming weak convergence)

$$\theta_2 = \lim_{n \to \infty} \frac{\log \|M_n(\omega)|_{\omega^\perp}\|}{n}$$

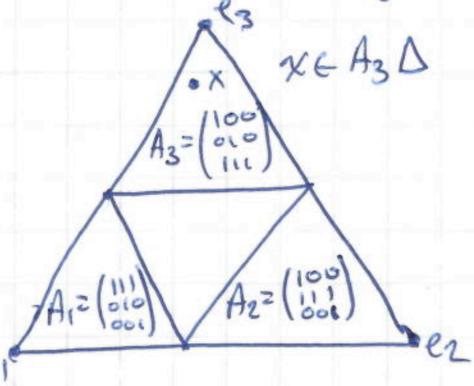
avec

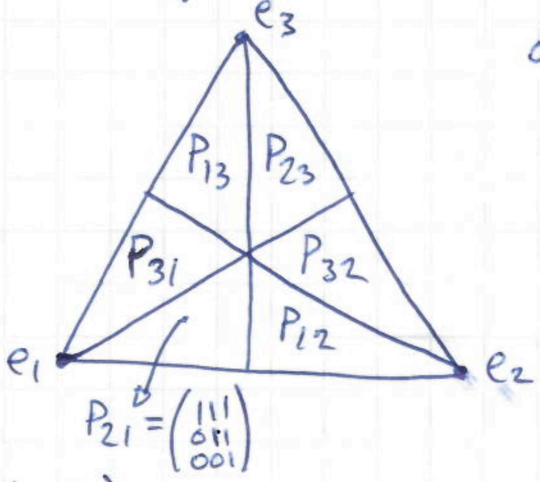
$$\|M_n(\omega)|_{\omega^\perp}\| = \sup_{v \in \omega^\perp} \frac{\|M_n(\omega) \cdot v\|}{\|v\|}$$



Arnoux-Rauzy Poincaré algorithm

Soit $\Delta = \{ (w_1, w_2, w_3) \in \mathbb{R}^3 \mid w_1 + w_2 + w_3 = 1 \}$



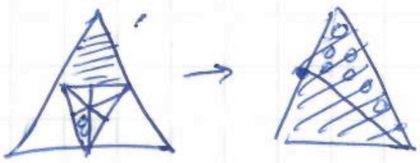


$\alpha_k : \begin{cases} i \mapsto ik \\ j \mapsto jk \\ k \mapsto k \end{cases}$

$\pi_{jk} : \begin{cases} i \mapsto ijk \\ j \mapsto jk \\ k \mapsto k \end{cases}$

$$M : \Delta \to GL(3, \mathbb{Z})$$
$$\vec{x} \mapsto \begin{cases} A_k, \text{ si } x \in A_k \Delta \text{ pour } k = 1, 2 \text{ ou } 3 \\ P_{jk}, \text{ sinon et si } x \in P_{jk}\Delta \text{ pour } \{i,j,k\} = \{1,2,3\} \end{cases}$$

Algo :
$$T : \Delta \to \Delta$$
$$\vec{x} \mapsto [M(\vec{x})]^{-1} \vec{x}$$



soustraire ...

Sequence of matrices associated to $\vec{x} \in \Delta$

$$M_0(\vec{x}) = Id \quad , \quad M_n(\vec{x}) = M(x) \cdot M(Tx) \cdot M(T^2 x) \cdots M(T^{n-1} x)$$

Substitutions

$$\sigma : \Delta \to \{\text{Substitutions}\}$$
$$\vec{x} \mapsto \begin{cases} \alpha_k \text{ si } \vec{x} \in A_k \Delta \\ \pi_{jk} \text{ sinon si } \vec{x} \in P_{jk} \Delta \end{cases}$$

S-adic word (discrete line)

$$W(\vec{x}) = \lim_{n \to \infty} \sigma(x) \cdot \sigma(Tx) \cdot \sigma(T^2 x) \cdots \sigma(T^{n-1} x)(1)$$

EXAMPLE

$\vec{x} = (\pi - 3, e - 2, 6 - e - \pi) \cdot 10^5$

$= (14159, 71828, 14013) \in A_2 \Delta$

$T\vec{x} = (14159, 43656, 14013) \in A_2 \Delta$

$T^2\vec{x} = (14159, 15484, 14013) \in P_{12} \Delta$

$T^3\vec{x} = (146, 1325, 14013) \in A_3 \Delta$

$T^{12}\vec{x} = (146, 1325, 774) \in A_2 \Delta$

$T^{13}\vec{x} = (146, 405, 774)$

etc...

$M_0(\vec{x}) = Id$

$M_1(\vec{x}) = A_2 = \begin{pmatrix} 1&0&0 \\ 1&1&1 \\ 0&0&1 \end{pmatrix}$

$M_2(x) = A_2 A_2 = \begin{pmatrix} 1&0&0 \\ 2&1&2 \\ 0&0&1 \end{pmatrix}$

$M_3(\vec{x}) = A_2 A_2 P_{12} = \begin{pmatrix} 1&1&1 \\ 2&3&5 \\ 0&0&1 \end{pmatrix}$

$M_4(\vec{x}) = A_2 A_2 P_{12} A_3 = \begin{pmatrix} 2&1&1 \\ 8&4&5 \\ 1&1&1 \end{pmatrix}$

$M_{12}(\vec{x}) = A_2 A_2 P_{13} A_3^9 = \begin{pmatrix} 10&9&1 \\ 48&44&5 \\ 9&9&1 \end{pmatrix}$

$M_{13}(\vec{x}) = A_2 A_2 P_{13} A_3^9 A_2 = \begin{pmatrix} 19&9&10 \\ 94&45&51 \\ 18&9&10 \end{pmatrix}$

...

$W_1(\vec{x}) = \alpha_2(1) = 12$

$W_2(\vec{x}) = \alpha_2 \alpha_2 (1) = 122$

$W_3(\vec{x}) = \alpha_2 \alpha_2 \pi_{12}(1) = 1222$

$W_4(\vec{x}) = \alpha_2 \alpha_2 \pi_{12} \alpha_3 (1) = 12223221222$

...

```
r"""
Computations of Lyapunov exponents.

This file illustrates pseudo code to compute lyapunov exponents.
It has been extracted from a larger file of several hundreds of lines I am
working on. The language used is Cython.

AUTHORS:

 - Vincent Delecroix, C code, Computation of Lyapounov exponents for Brun
   algorithm, June 2013.
 - Sebastien Labbe, Invariant measures, Lyapounov exponents and natural
   extensions for a dozen of algorithms, October 2013.
"""
cdef struct PairPoint3d:
    double x
    double y
    double z
    double u
    double v
    double w

def lyapounov_exponents(self, algo, int n_iterations=1000, int step=16):
    r"""
    This returns the lyapounov exponents of some algorithms.

    INPUT:

    - ``algo`` -- string, the algorithm to consider
    - ``n_iterations`` -- integer
    - ``step`` -- integer, do the sum of lyapounov every n step
      (between 1 and 30 seems good values, 16 seems the fastest while
      still being good for additive algorithms)

    OUTPUT:

    tuple : (theta1, theta2, theta2/theta1)

    NOTE:: the code of this method was translated from C to cython. The
    C version is from Vincent Delecroix.

    EXAMPLES::

        sage: lyapounov_exponents('brun', 1000000, step=16)
        (0.3049429393152174, -0.1120652699014143, -0.367495867105725)

    """
    cdef double theta1=0, theta2=0    # values of Lyapunov exponents
    cdef double theta1c=0, theta2c=0  # compensation (for Kahan summation algorithm)
    cdef double x,y,z           # vector (x,y,z)
    cdef double u,v,w           # vector (u,v,w)
    cdef double p,s,t           # temporary variables
    cdef unsigned int i         # loop counter

    # random initial values
    x = random(); y = random(); z = random();
    u = random() - .5; v = random() - .5; w = random() - .5;

    # Order (x,y,z)
    if y > z: z,y = y,z
    if x > z: x,y,z = y,z,x
    elif x > y: x,y = y,x

    # Normalize (x,y,z)
    s = x + y + z
    x /= s; y /= s; z /= s

    # Gram Shmidtt on (u,v,w)
    p = x*u + y*v + z*w
    s = x*x + y*y + z*z
```

```
u -= p*x/s; v -= p*y/s; w -= p*z/s

# Normalize (u,v,w)
s = abs(u) + abs(v) + abs(w);
u /= s; v /= s; w /= s

cdef PairPoint3d P
P.x = x
P.y = y
P.z = z
P.u = u
P.v = v
P.w = w

# Loop
for i from 0 <= i < n_iterations:

    # Apply Algo. i.e.
    # (x,y,z) = A^{-1} (x,y,z)
    # (u,v,w) = A^T    (u,v,w)
    P = Apply_algo(P, algo)

    # Save some computations
    if i % step == 0:

        # Sum the first lyapounov exponent
        s = P.x + P.y + P.z
        p = -log(s) - theta1c
        t = theta1 + p
        theta1c = (t-theta1) - p   # mathematically 0 but not for a computer!!
        theta1 = t
        P.x /= s; P.y /= s; P.z /= s;

        # Sum the second lyapounov exponent
        s = abs(P.u) + abs(P.v) + abs(P.w)
        p = log(s) - theta2c
        t = theta2 + p
        theta2c = (t-theta2) - p   # mathematically 0 but not for a computer!!
        theta2 = t

        # the following gramm shimdts seems to be useless, but it is not!!!
        p = P.x*P.u + P.y*P.v + P.z*P.w
        s = P.x*P.x + P.y*P.y + P.z*P.z
        P.u -= p*P.x/s; P.v -= p*P.y/s; P.w -= p*P.z/s
        s = abs(P.u) + abs(P.v) + abs(P.w)
        P.u /= s; P.v /= s; P.w /= s

return theta1/n_iterations, theta2/n_iterations, theta2/theta1
```