Takagi02

October 22, 2020

1 Les bosses de la fonction de Takagi

Marc Lorenzi

22 octobre 2020

[1]:
```
import matplotlib.pyplot as plt
import math
from fractions import Fraction
```

[2]:
```
plt.rcParams['figure.figsize'] = (10, 6)
```

Nous allons nous intéresser dans ce notebook aux **bosses** de la fonction de Takagi, définie pour tout réel x par

$$\tau(x) = \sum_{k=0}^{\infty} \frac{1}{2^k} \ll 2^k x \gg$$

où $\ll x \gg$ désigne la distance de x à $\mathbb{Z}$. On pose également, pour tout $n \in \mathbb{N}$,

$$\tau_n(x) = \sum_{k=0}^{n} \frac{1}{2^k} \ll 2^k x \gg$$

Le notebook `Takagi01` détaille un certain nombre de propriétés de ces fonctions. En particulier, les fonctions τ_n tendent uniformément vers τ sur $\mathbb{R}$ lorsque n tend vers l'infini. La fonction τ est continue sur $\mathbb{R}$ et dérivable nulle part.

[3]:
```
def alpha(x): return math.floor(x + 1/2)

def delta(x): return abs(x - alpha(x))

def tau(n, x):
    return sum([delta(2 ** k * x) / 2 ** k for k in range(n + 1)])
```

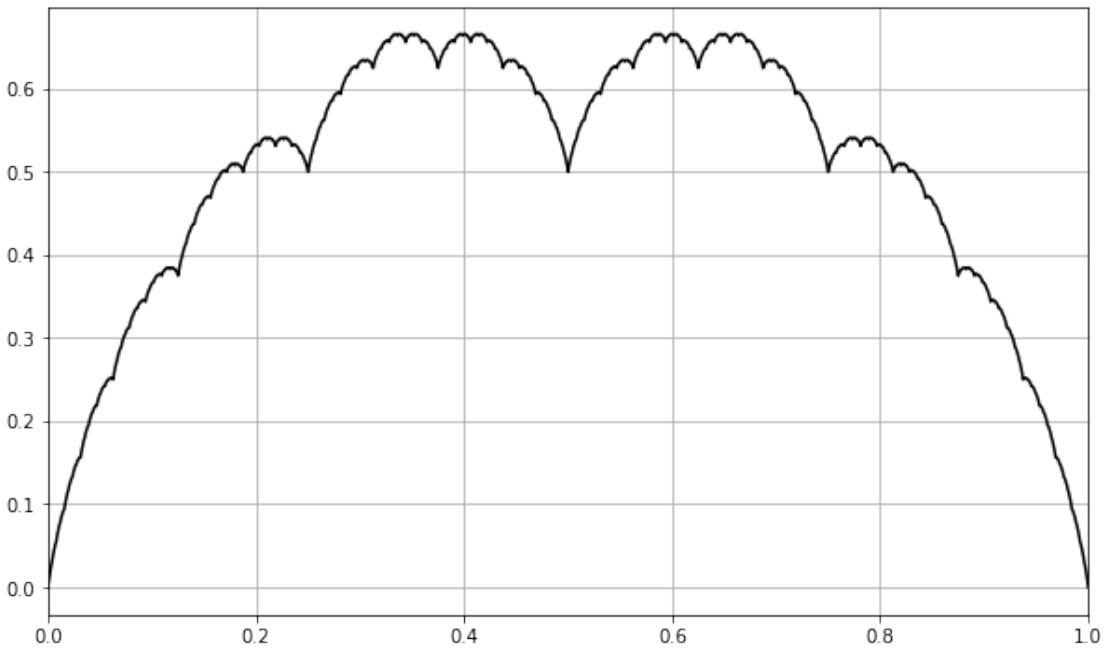
La fonction `plot_fun` trace le graphe de la fonction f sur le segment $[a, b]$. Les paramètres `dots` , `cadres` et `pentes` seront utilisés plus loin.

[4]:
```
def subdi(a, b, n):
    d = (b - a) / n
    return [a + k * d for k in range(n + 1)]
```

[5]:
```
def plot_fun(f, a, b, n, cadres=[], dots=False, pentes=False):
    xs = subdi(a, b, n)
    ys = [f(x) for x in xs]
    plt.xlim(a, b)
    plt.plot(xs, ys, 'k')
    if dots: plt.plot(xs, ys, 'or')
    if pentes:
        for k in range(n):
            delta = int((f(xs[k + 1]) - f(xs[k])) / (xs[k + 1] - xs[k]))
            x = (0.6 * xs[k + 1] + 0.4 * xs[k])
            y = (f(xs[k + 1]) + f(xs[k])) / 2
            plt.text(x, y, str(delta))
    for a, b in cadres:
        us = [a, b, b, a, a]
        vs = [f(a), f(a), f(a) + 2/3 * (b - a), f(a) + 2/3 * (b - a), f(a)]
        plt.plot(us, vs, 'r')
    plt.grid()
```

Voici le graphe de τ_{10}, qui est une approximation du graphe de τ.

[6]:
```
plot_fun(lambda x: tau(10, x), 0, 1, 2 ** 10)
```



On remarque sur le graphique qu'un certain nombre de parties du graphe forment des « bosses » qui ont l'air « semblables » au graphe tout entier. Par exemple, la partie corresondant à $\frac{1}{4} \leq x \leq \frac{1}{2}$. Nous allons voir que les guillemets peuvent être enlevés, et que ces parties sont effectivement les images du graphe entier par des similitudes (en fait des homothéties).

1.1 1. Pentes de τ_n.

1.1.1 1.1 Développement dyadique d'un réel

Tout réel $x \in [0, 1]$ peut s'écrire sous la forme

$$x = \sum_{k=1}^{\infty} \frac{b_k}{2^k} = 0.b_1 b_2 \dots$$

où les $b_k \in \{0, 1\}$ sont les chiffres de l'écriture binaire de x. Un tel développement est unique, sauf lorsque $x \in \mathbb{D} \cap]0, 1[$ est un nombre dyadique. Dans ce cas, x possède deux développements :

- Un développement où il existe un entier $n_0 \geq 1$ tel que $b_{n_0-1} = 1$ et pour tout $k \geq n_0$, $b_k = 0$.
- Un développement où $b_{n_0-1} = 0$ et pour tout $k \geq n_0$, $b_k = 1$.

Les réels 0 et 1 sont un peu à part. Ils possèdent un unique développement dyadique :

$$0 = 0.000 \dots \text{ et } 1 = 0.111 \dots$$

La fonction `chiffres_binaires` prend en paramètre un réel $x \in [0, 1[$ et un entier n. Elle renvoie la liste des n premiers chiffres du développement dyadique de x.

[7]:
```
def chiffres_binaires(x, n):
    s = []
    for k in range(n):
        x = 2 * x
        s.append(math.floor(x))
        x = x - math.floor(x)
    return s
```

[11]:
```
for k in range(16):
    print('%3d%20s' % (k, chiffres_binaires(Fraction(k, 16), 4)))
```

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

```
 9        [1, 0, 0, 1]
10        [1, 0, 1, 0]
11        [1, 0, 1, 1]
12        [1, 1, 0, 0]
13        [1, 1, 0, 1]
14        [1, 1, 1, 0]
15        [1, 1, 1, 1]
```

Notons, pour $b \in \{0,1\}$, $\overline{b} = 1 - b$. Soit $x = 0.b_1b_2\ldots \in [0,1]$.

- Si $b_1 = 0$, $\ll x \gg = x = 0.b_1b_2\ldots$.
- Si $b_1 = 1$, $\ll x \gg = 1 - x = 0.\overline{b}_1\overline{b}_2\ldots$.

Remarquons que si x est un nombre dyadique et possède donc en général deux développements, les formules ci-dessus donnent l'un ou l'autre des développements selon les cas. Par exemple,

- $\frac{1}{2} = 0.100\ldots$ donne $\ll \frac{1}{2} \gg = 0.011\ldots = \frac{1}{2}$.
- $\frac{1}{2} = 0.011\ldots$ donne $\ll \frac{1}{2} \gg = 0.011\ldots = \frac{1}{2}$.
- $1 = 0.111\ldots$ donne $\ll 1 \gg = 0.000\ldots = 0$.

Plus généralement, pour tout $n \geq 0$,

- Si $b_{n+1} = 0$, $\ll 2^n x \gg = 0.b_{n+1}b_{n+2}\ldots$.
- Si $b_{n+1} = 1$, $\ll 2^n x \gg = 0.\overline{b}_{n+1}\overline{b}_{n+2}\ldots$.

Notons enfin, pour tout $n \in \mathbb{N}$,

$$D_n(x) = \sum_{k=1}^{n} (-1)^{b_k}$$

$D_n(x)$ est égal au nombre de 0 moins le nombre de 1 dans les n premiers chiffres du développement dyadique de x.

La fonction `somme_alt` prend en paramètre un nombre dyadique x et un entier n. Elle renvoie $D_n(x)$.

[12]:
```
def somme_alt(x, n):
    bs = chiffres_binaires(x, n)
    s = 0
    for x in bs:
        if x == 0: s = s + 1
        else: s = s - 1
    return s
```

[14]:
```
for k in range(16):
    x = Fraction(k, 16)
    cb = chiffres_binaires(x, 4)
    print('%3d%20s%20s' % (k, cb, [somme_alt(x, j) for j in range(1, 5)]))
```

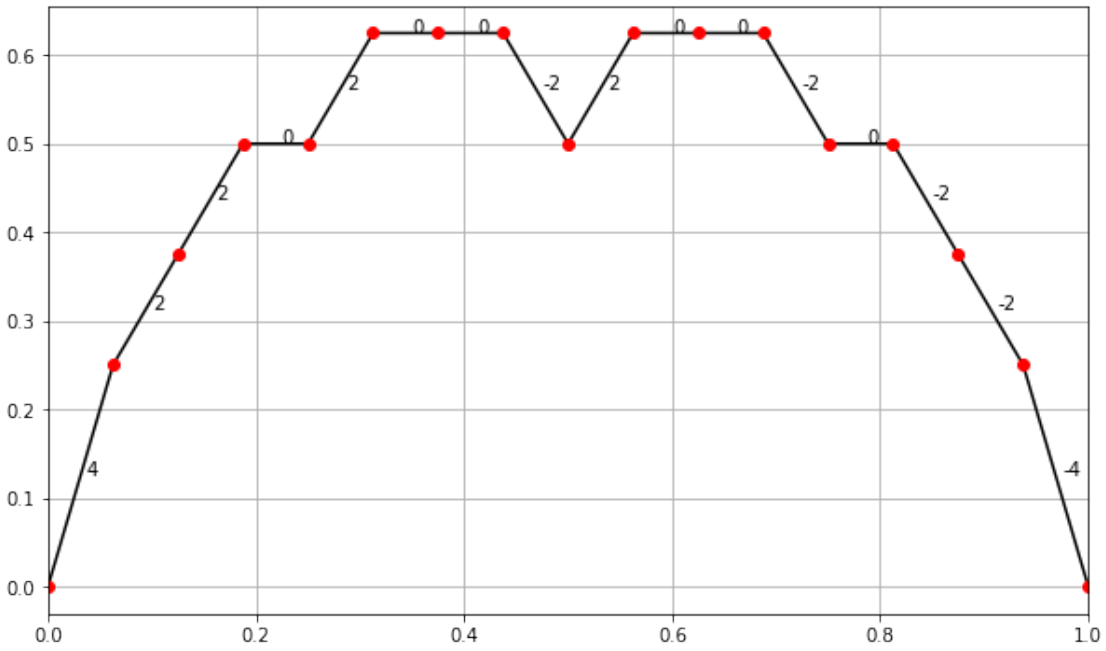
```
 0        [0, 0, 0, 0]        [1, 2, 3, 4]
 1        [0, 0, 0, 1]        [1, 2, 3, 2]
```

2	[0, 0, 1, 0]	[1, 2, 1, 2]
3	[0, 0, 1, 1]	[1, 2, 1, 0]
4	[0, 1, 0, 0]	[1, 0, 1, 2]
5	[0, 1, 0, 1]	[1, 0, 1, 0]
6	[0, 1, 1, 0]	[1, 0, -1, 0]
7	[0, 1, 1, 1]	[1, 0, -1, -2]
8	[1, 0, 0, 0]	[-1, 0, 1, 2]
9	[1, 0, 0, 1]	[-1, 0, 1, 0]
10	[1, 0, 1, 0]	[-1, 0, -1, 0]
11	[1, 0, 1, 1]	[-1, 0, -1, -2]
12	[1, 1, 0, 0]	[-1, -2, -1, 0]
13	[1, 1, 0, 1]	[-1, -2, -1, -2]
14	[1, 1, 1, 0]	[-1, -2, -3, -2]
15	[1, 1, 1, 1]	[-1, -2, -3, -4]

1.1.2 1.2 Les pentes de τ_n

Voici le graphe de τ_3 avec les pentes indiquées sur chacun des segments de la courbe.

[15]:
```
n = 3
plot_fun(lambda x: tau(n, x), 0, 1, 2 ** (n + 1), dots=True, pentes=True)
```



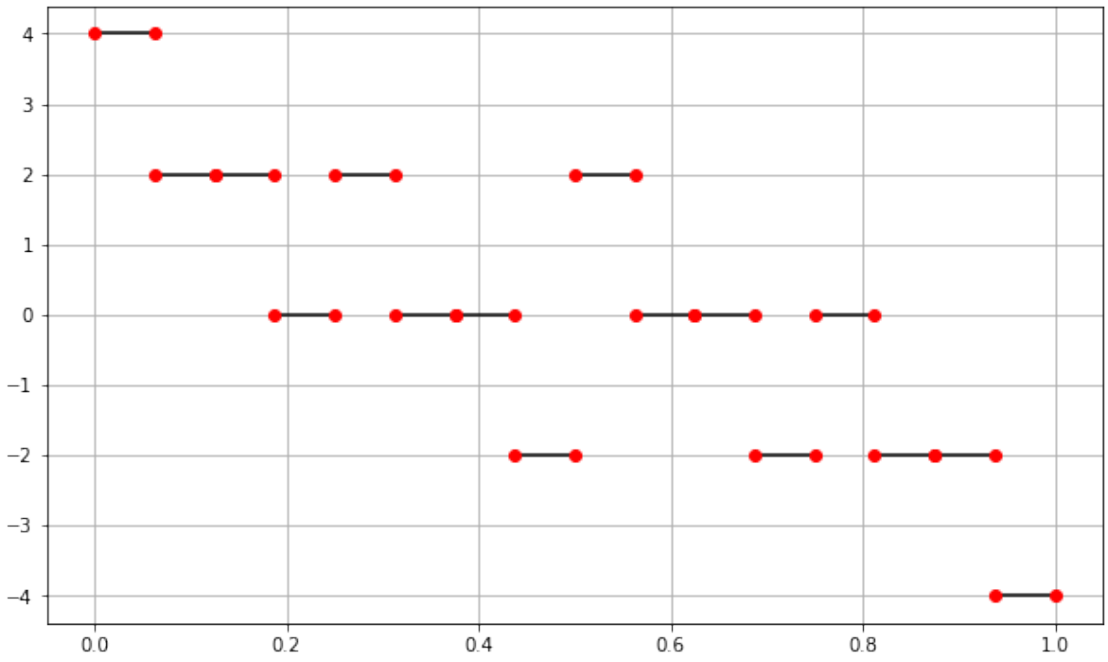
Le graphe de τ_3 est constitué de $16 = 2^4$ segments.

La fonction `plot_D` trace le graphe de D_n.

[16]:
```
def plot_D(n):
    N = 2 ** n
    xs = [Fraction(k, N) for k in range(N + 1)]
    for k in range(N):
        y = somme_alt(xs[k], n)
        plt.plot([float(xs[k]), float(xs[k + 1])], [y, y], 'k')
        plt.plot([float(xs[k]), float(xs[k + 1])], [y, y], 'or')
    plt.grid()
```

Voici le graphe de D_4.

[17]:
```
plot_D(4)
```



Bien entendu ce n'est pas une coïncidence si les valeurs de D_4, constante par morceaux, sont égales aux pentes aux pentes de τ_3.

Proposition. Soit $n \geq 0$.

- La fonction τ_n est affine sur les intervalles $[\frac{p}{2^{n+1}}, \frac{p+1}{2^{n+1}}]$, $0 \leq p < 2^{n+1}$.
- La fonction D_{n+1} est constante sur les intervalles $[\frac{p}{2^{n+1}}, \frac{p+1}{2^{n+1}}[$.
- La pente de τ_n sur chacun des intervalles est égale à la valeur de la fonction D_{n+1} sur l'intervalle.

Démonstration. Posons $u = 0.b_1 \dots b_{n+1}00\dots$ où les b_k valent 0 ou 1. Pour tout $x \in [u, u + \frac{1}{2^{n+1}}[$, $x = 0.b_1 b_2 \dots$ où les b_k coincident avec ceux de u jusqu'à l'indice $n+1$. La fonction D_{n+1} est donc constante sur l'intervalle $[u, v[$.

Soit $0 \leq k \leq n$. On a par périodicité de $\ll \gg$,

$$\ll 2^k x \gg = \ll b_1 \dots b_k . b_{k+1} \dots \gg = \ll 0.b_{k+1} b_{k+2} \dots \gg$$

Ceci est en particulier valable si $x = u$. Deux cas se présentent alors :

- Si $b_{k+1} = 0$, on a $\ll 2^k x \gg = 0.b_{k+1} b_{k+2} \dots = \ll 2^k u \gg + 2^k (x - u)$.
- Si $b_{k+1} = 1$, on a $\ll 2^k x \gg = 1 - 0.b_{k+1} b_{k+2} \dots = 0.\bar{b}_{k+1} \bar{b}_{k+2} \dots = \ll 2^k u \gg - 2^k (x - u)$.

Ces deux cas peuvent être rassemblés en constatant que

$$\ll 2^k x \gg = \ll 2^k u \gg + (-1)^{b_{k+1}} 2^k (x - u)$$

ou encore

$$\frac{1}{2^k} \ll 2^k x \gg = \frac{1}{2^k} \ll 2^k u \gg + (-1)^{b_{k+1}} (x - u)$$

Sommons pour k allant de 0 à n. On obtient

$$\tau_n(x) = \tau_n(u) + (x - u) \sum_{k=0}^{n} (-1)^{b_{k+1}} = \tau_n(u) + D_{n+1}(u)(x - u)$$

Remarque. Remarquons que la pente maximale de la fonction τ_n est obtenue pour $u = 0.00 \dots 0 = 0$, et vaut $n + 1$. La pente minimale est quant à elle obtenue pour $u = 0.11 \dots 1 = 1 - \frac{1}{2^{n+1}}$, et vaut $-(n + 1)$.

1.1.3 1.3 Nombres dyadiques équilibrés

Définition. Soit $m \in \mathbb{N}*$. $x = 0.b_1 \dots b_{2m} \in \mathbb{D} \cap [0, 1]$.

- On dit que x est **équilibré d'ordre** m lorsque $D_{2m}(x) = 0$.
- S'il y a n indices $1 \leq j \leq 2m$ tels que $D_j(x) = 0$, on dit que x est de **génération** n.
- Si $D_j(x) \geq 0$ pour tout $j \leq 2m$, on dit que x est **dominant**.

On convient également que 0 est équilibré d'ordre 0.

La fonction `equilibre` prend un nombre dyadique x et un entier n en paramètres. Elle renvoie `True` si $D_n(x) = 0$, et `False` sinon.

[18]:
```
def equilibre(x, n):
    return somme_alt(x, n) == 0
```

[19]:
```
m = 3
N = 2 ** (2 * m)
s = [Fraction(k, N) for k in range(N)]
for x in s:
    if equilibre(x, 2 * m): print(x)
```

```
7/64
11/64
13/64
7/32
19/64
21/64
11/32
25/64
13/32
7/16
35/64
37/64
19/32
41/64
21/32
11/16
49/64
25/32
13/16
7/8
```

Écrivons une fonction `caract` prenant en paramètre un nombre dyadique x et un entier m et renvoyant les caractéristiques de x sous forme d'un triplet (e, g, d), où

- e est `True` si et seulement si x est équilibré d'ordre $2m$. Et dans le cas où e vaut `True` :
- g est la génération de x.
- d vaut `True` si et seulement si x est dominant.

[20]:
```
def caract(x, m):
    g = 0
    d = True
    bs = chiffres_binaires(x, m)
    s = 0
    for b in bs:
        if b == 0: s = s + 1
        else: s = s - 1
        if s == 0: g = g + 1
        if s < 0: d = False
    if s == 0: return (True, g, d)
    else: return (False, g, d)
```

[21]:
```
m = 3
N = 2 ** (2 * m)
s = [Fraction(k, N) for k in range(N)]
print('%10s|%5s|%9s' % ('x', 'géné', 'dominant'))
for x in s:
    e, g, d = caract(x, 2 * m)
    if e:
```

```
        print('%10s|%5d|%9s' % (x, g, d))
```

```
         x| géné| dominant
      7/64|    1|     True
     11/64|    1|     True
     13/64|    2|     True
      7/32|    2|    False
     19/64|    2|     True
     21/64|    3|     True
     11/32|    3|    False
     25/64|    3|    False
     13/32|    3|    False
      7/16|    2|    False
     35/64|    2|    False
     37/64|    3|    False
     19/32|    3|    False
     41/64|    3|    False
     21/32|    3|    False
     11/16|    2|    False
     49/64|    2|    False
     25/32|    2|    False
     13/16|    1|    False
       7/8|    1|    False
```

Remarque. Nous n'expliquerons pas, par manque de place, en quoi le caractère dominant ou la génration du nombre dyadique équilibré x ont une importance.

1.2 2. Homothéties

1.2.1 2.1 Le théorème

Proposition. Soit $m \in \mathbb{N}$. Soit $u = \frac{p}{2^{2m}}$ où $0 \leq p < 2^{2m}$ un nombre dyadique équilibré, où p est impair. Alors, pour tout $x \in [u, u + \frac{1}{2^{2m}}]$, on a

$$\tau(x) = \tau(u) + \frac{1}{2^{2m}}\tau(2^{2m}(x-u))$$

Démonstration. Posons $x = u + t$ où $0 \leq t < \frac{1}{2^{2m}}$. On a

$$\tau(x) = \tau(u+t) = \tau_{2m-1}(u+t) + \sum_{k=2m}^{\infty} \frac{1}{2^k} \ll 2^k t \gg$$

car pour $k \geq 2m$, $2^k u \in \mathbb{N}$. Posant $k = 2^{2m} + j$ dans la somme, il vient

$$\sum_{k=2m}^{\infty} \frac{1}{2^k} \ll 2^k t \gg = \frac{1}{2^{2m}} \sum_{j=0}^{\infty} \frac{1}{2^j} \ll 2^j 2^{2m} t \gg = \frac{1}{2^{2m}}\tau(2^{2m}t)$$

Par ailleurs, $\tau_{2m-1}(u) = \tau(u)$ car u est dyadique, de dénominateur 2^{2m}. Ainsi,

$$\tau(x) = A + \tau(u) + \frac{1}{2^{2m}}\tau(2^{2m}(x-u))$$

où

$$A = \tau_{2m-1}(x) - \tau_{2m-1}(u)$$

rappelons-nous que τ_{2m-1} est affine sur le segment $[u, v]$. Sa pente est $D_{2m}(u) = 0$, et donc τ_{2m-1} est contante. Ainsi, $A = 0$.

1.2.2 2.2 Conséquences sur le graphe

Pour $0 \leq u < v \leq 1$, notons $\mathcal{G}_{u,v}$ le graphe de la restriction de τ à l'intervalle $[u, v]$. Notons également $\mathcal{G} = \mathcal{G}_{0,1}$.

Soit $(x, y) \in \mathcal{G}_{u,v}$. On a

$$2^{2m}(y - \tau(u)) = \tau(2^{2m}(x-u))$$

Soit $h : (x, y) \mapsto 2^{2m}(x - u, y - \tau(u))$. La fonction h est l'homothétie de centre $(u, \tau(u))$ et de rapport 2^{2m}. On a $h(x, y) \in \mathcal{G}$. Ainsi, $h(\mathcal{G}_{u,v}) \subset \mathcal{G}$. L'inclusion réciproque est laissée au lecteur. On a donc

$$h(\mathcal{G}_{u,v}) = \mathcal{G}$$

Ainsi, $\mathcal{G}_{u,v}$ **est l'image de** $\mathcal{G}$ **par l'homothétie** h^{-1}**.**

1.2.3 2.3 Bosses

La fonction `bosses` prend en paramètre un entier m. Posons $N = 2^{2m}$. La fonction renvoie la liste des intervalles $[\frac{k}{N}, \frac{k+1}{N}]$ où $0 \leq k < N$ et $D_{2m}(\frac{k}{N}) = 0$.

[22]:
```
def bosses(m):
    N = 2 ** (2 * m)
    s = [Fraction(k, N) for k in range(N) if equilibre(Fraction(k, N), 2 * m)]
    return [(x, x + Fraction(1, N)) for x in s]
```

[26]:
```
print(bosses(1))
```

```
[(Fraction(1, 4), Fraction(1, 2)), (Fraction(1, 2), Fraction(3, 4))]
```

[27]:
```
print(bosses(2))
```

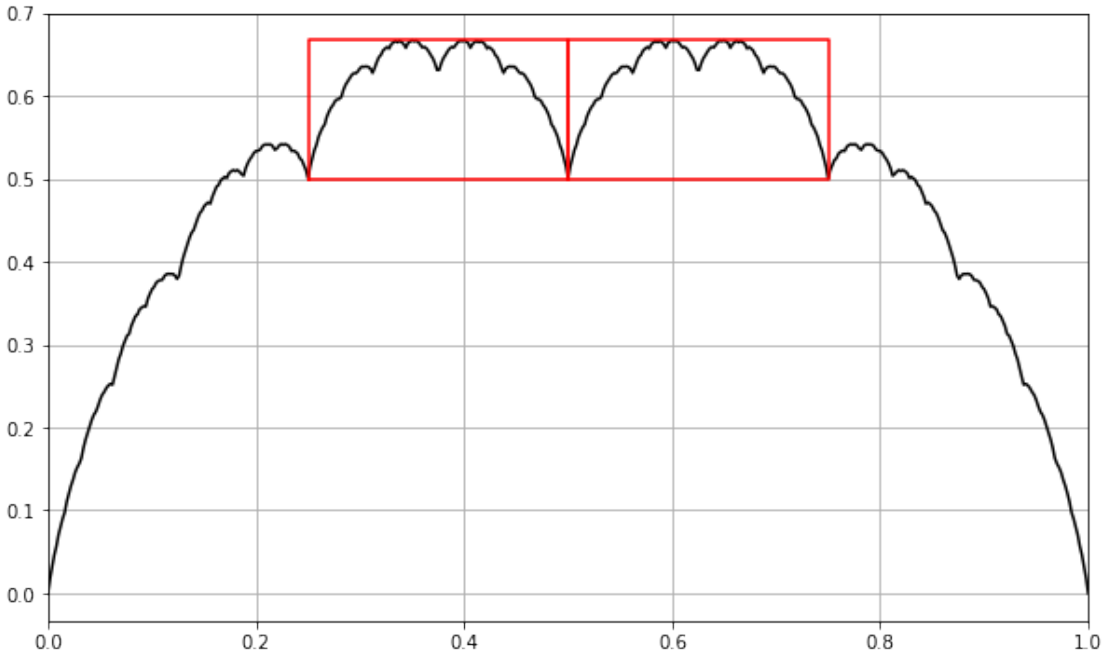
```
[(Fraction(3, 16), Fraction(1, 4)), (Fraction(5, 16), Fraction(3, 8)),
(Fraction(3, 8), Fraction(7, 16)), (Fraction(9, 16), Fraction(5, 8)),
(Fraction(5, 8), Fraction(11, 16)), (Fraction(3, 4), Fraction(13, 16))]
```

1.2.4 2.4 Tracés

[28]:
```
def trace_bosses(m):
    bss = []
    for k in range(1, m + 1):
        bss = bss + bosses(k)
    plot_fun(lambda x: tau(30, x), 0, 1, 500, cadres=bss)
```

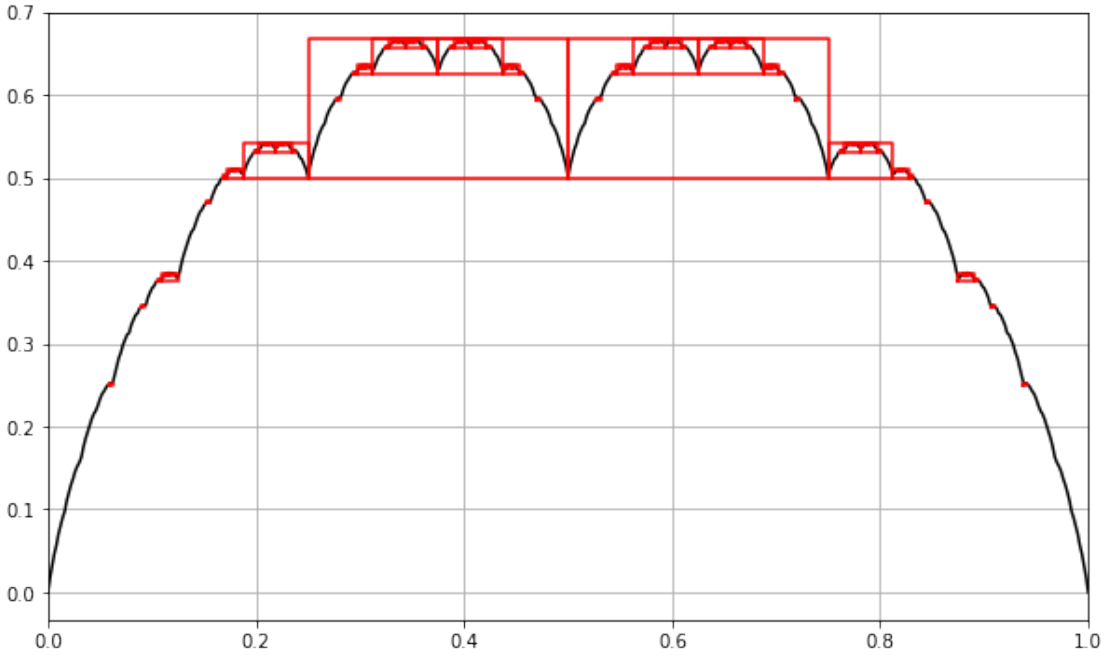
Voici les bosses d'ordre 1.

[29]:
```
trace_bosses(1)
```



Et voici les bosses d'ordre inférieur ou égal à 4.

[30]:
```
trace_bosses(4)
```



1.2.5 2.5 Extrema locaux

Voici deux théorèmes énoncés sans preuve.

Proposition [Maxima locaux]. Soit $x = \sum_{k=1}^{\infty} \frac{b_k}{2^k}$. La fonction de Takagi a un maximum local en x si et seulement si $(x, \tau(x))$ est au sommet d'une bosse, ce qui équivaut à l'existence d'un entier $m \in \mathbb{N}$ tel que

- $b_1 + \ldots + b_{2m} = m$.
- Pour tout $n > m$, $b_{2n-1} + b_{2n} = 1$.

Corollaire. L'ensemble des réels en lesquels τ a un maximum local est dense dans $\mathbb{R}$.

Proposition [Minima locaux]. L'ensemble des réels en lesquels τ a un minimum local est l'ensemble $\mathbb{D}$ des nombres dyadiques.

Corollaire. L'ensemble des réels en lesquels τ a un minimum local est dense dans $\mathbb{R}$.

1.3 3. Compter les bosses

1.3.1 3.1 Combien de bosses ?

Combien y a-t-il de bosses d'ordre donné ? de génération donnée ? Pour tout entier naturel n, notons C_n le $n$ième nombre de Catalan :

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Proposition. Soit $m \in \mathbb{N}^*$.

- Il y a $\binom{2m}{m}$ bosses d'ordre m.
- Il y a C_m bosses dominantes d'ordre m.
- Il y a $2C_{m-1}$ bosses de génération 1 d'ordre m.

Démonstration.

- Il s'agit de compter les $2m$-uplets $(b_1, \ldots, b_{2m})$ correspondant à des nombres dyadiques équilibrés. Ce sont ceux pour lesquels il y a m chiffres b_k égaux à 0 (et m égaux à 1). Il y a $\binom{2m}{m}$ tels nombres (choix des b_k égaux à 0).
- Associons au chiifre 0 le symbole de parenthèse ouvrante, et au chiffre 1 le symbole de parenthèse fermante. Les nombres équilibrés d'ordre m dominants sont alors associés aux parenthésages bien formés contenant m parenthèses ouvrantes et m parenthèses fermantes. Par exemple, (()())() correspond au nombre 0.00101101. Il est « connu » que le nombre de tels parenthésages est le $m$ième nombre de Catalan C_m.
- x est de génération 1 si et seulement si il existe un unique indice $j \leq 2m$ tel que $D_j(x) = 0$. Comme $D_{2m}(x) = 0$, cet indice est $j = 2m$. Toujours en raisonnant sur des parenthésages, nous recherchons les mots de longueur $2m$ de la forme
  - (u) où u est un parenthésage bien formé de longueur $2m-2$: il y en a C_{m-1}.
  - ou alors $)u'($, où u' est le « conjugué » d'un parenthésage bien formé (obtenu par l'échange des parenthèses ouvrante et fermante). Il y en a aussi C_{m-1}.

D'où le résultat.

Voici quelques fonctions permettant le calcul des coefficients binomiaux et des nombres de Catalan.

[31]:
```
def power_down(n, k):
    p = 1
    for j in range(k): p = p * (n - j)
    return p
```

[32]:
```
def binomial(n, k):
    return power_down(n, k) // power_down(k, k)
```

[33]:
```
def catalan(n):
    return binomial(2 * n, n) // (n + 1)
```

[34]:
```
print([catalan(k) for k in range(10)])
```

```
[1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862]
```

La fonction `compte_theorique` prend un entier m en paramètre. Elle renvoie le triplet (nombre de bosses d'ordre m, nombre de bosses dominantes d'ordre m, nombre de bosses d'ordre m de génération 1).

[35]:
```
def compte_theorique(m):
    return (binomial(2 * m, m), catalan(m), 2 * catalan(m - 1))
```

[36]:
```
for k in range(1, 20):
    print(k, compte_theorique(k))
```

```
1 (2, 1, 2)
2 (6, 2, 2)
3 (20, 5, 4)
4 (70, 14, 10)
5 (252, 42, 28)
6 (924, 132, 84)
7 (3432, 429, 264)
8 (12870, 1430, 858)
9 (48620, 4862, 2860)
10 (184756, 16796, 9724)
11 (705432, 58786, 33592)
12 (2704156, 208012, 117572)
13 (10400600, 742900, 416024)
14 (40116600, 2674440, 1485800)
15 (155117520, 9694845, 5348880)
16 (601080390, 35357670, 19389690)
17 (2333606220, 129644790, 70715340)
18 (9075135300, 477638700, 259289580)
19 (35345263800, 1767263190, 955277400)
```

Voici le nombre de bosses d'ordre 1000. Il y en a environ 10^{600}.

[37]:
```
nb = compte_theorique(1000)[0]
print('nombre de chiffres : ', len(str(nb)))
print(nb)
```

```
nombre de chiffres :  601
204815162698948971433516250298082504439642488798139703382038263767174818620208375582893299418261020620146476631999802369241548179800452479201804754976926157856301289663432064714851152395251651227768588611539546256147907378668464154444533617613770073855673814589630071306510455959514479888746206368718514551828551173166276253663773084682932255389049743859481431755030783796444370810085163724827462791417016619883764840843541430817785947037746565188475514680749694674923803033101818723298009668567458560252549910118113525353465888794196665367490451130611009631190627034250229315591110897673396399114912 0
```

1.3.2 3.2 Tests

Mettons nos fonctions à l'épreuve de la théorie. La fonction `test_bosses` prend un entier m en paramètre. Elle renvoie un triplet dont les composantes sont le nombre de bosses d'ordre m, le nombre de bosses dominantes d'ordre m et le nombre de bosses de génération 1 d'ordre m. Ces nombres sont obtenus par des appels à la fonction `caract`.

[38]:
```
def test_bosses(m):
    N = 2 ** (2 * m)
    s = [Fraction(k, N) for k in range(N)]
    nb_bosses = 0
    nb_bosses_dominantes = 0
    nb_bosses_generation1 = 0
    for x in s:
        e, g, d = caract(x, 2 * m)
        if e:
            nb_bosses += 1
            if d: nb_bosses_dominantes += 1
            if g == 1: nb_bosses_generation1 += 1
    return (nb_bosses, nb_bosses_dominantes, nb_bosses_generation1)
```

[39]:
```
print(test_bosses(7))
```

```
(3432, 429, 264)
```

Que nous dit la théorie ?

[40]:
```
m = 7
print(compte_theorique(7))
```

```
(3432, 429, 264)
```

Nous avons tout bon .

1.3.3 3.3 Et après ?

L'étude des bosses est une première étape vers l'étude des **ensembles de niveau** de la fonction de Takagi. Pour tout $y \in [0, \frac{2}{3}]$, l'ensemble de niveau de y est

$$L(y) = \tau^{-1}(\{y\}) = \{x \in [0,1], \tau(x) = y\}$$

Cet ensemble peut être fini ou infini. Lorsqu'il est infini, il peut être dénombrable ou pas. Exemples :

- Dans *The Art of Computer Programming, Vol. 4*, D.E. Knuth signale dans un exercice que
$$L(\frac{1}{5}) = \{x, 1-x\}$$
où $x = \frac{83581}{87040}$. Le lecteur pourra vérifier que $\tau(x) = \frac{1}{5}$ à l'aide du notebook `Takagi01`.
- $L(\frac{1}{2})$ est infini dénombrable.
- $L(\frac{2}{3})$ a une mesure de Hausdorff strictement positive, et est donc un ensemble non dénombrable : l'ensemble des réels de $[0,1]$ où τ atteint son maximum n'est pas dénombrable.

L'étude des ensembles de niveau de τ est à lui seul un sujet de recherche. Nous nous contenterons en conclusion de ce notebook de donner le résultat suivant. Il devrait suffire à nous plonger dans des abîmes de perplexité.

Proposition.

- L'ensemble des y tels que $L(y)$ est fini est dense dans $[0, \frac{2}{3}]$.
- L'ensemble des y tels que $L(y)$ est infini dénombrable est dense dans $[0, \frac{2}{3}]$.
- L'ensemble des y tels que $L(y)$ est infini non dénombrable est dense dans $[0, \frac{2}{3}]$.

1.4 Références

- P. Allaart, K. Kawamura, The Takagi Function: a Survey (2011)
- Y. Baba, On Maxima of Takagi-Van der Waerden Functions (1984)