Bernoulli_DM

March 11, 2024

1 Polynômes et nombres de Bernoulli

Marc Lorenzi

11 mars 2024

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

1.1 1. Introduction

Définition. Soit $(B_n)_{n\in\mathbb{N}}$ la suite de polynômes définie par $B_0 = 1$ et pour tout $n \geq 1$,

$$B_n' = B_{n-1}$$

$$\int_0^1 B_n(x)\,dx = 0$$

B_n est le $n$ième polynôme de Bernoulli. Le réel $b_n = n!\,B_n(0)$ est le $n$ième nombre de Bernoulli.

Les « vrais » polynômes de Bernoulli sont, avec les notations du devoir maison, les polynômes $n!\,B_n$. Nous les appellerons les polynômes de Bernoulli-Wikipedia, pour que ceux qui se seraient (hélas) « inspirés » de ce que l'on peut trouver sur le net au lieu d'essayer de trouver par eux-mêmes puissent recoller à l'énoncé du devoir. Nous écrirons parfois dans la suite deux fonctions, une pour les polynômes de Bernoulli du DM, et une pour les polynômes de Bernoulli-Wikipedia.

Proposition. Pour tout $n \in \mathbb{N}$,

$$B_n = \frac{1}{n!}\sum_{k=0}^{n}\binom{n}{k} b_k X^{n-k}$$

Démonstration. On a $B_0 = 1 = b_0$. Soit $n \in \mathbb{N}$. Supposons que B_n a la forme voulue. B_{n+1} est égal à $n+1$ fois une primitive de B_n, donc

$$\begin{aligned} B_{n+1} &= B_{n+1}(0) + \frac{1}{n!}\sum_{k=0}^{n} \frac{1}{n+1-k}\binom{n}{k} b_k X^{n+1-k} \\ &= B_{n+1}(0) + \frac{1}{(n+1)!}\sum_{k=0}^{n} \frac{n+1}{n+1-k}\binom{n}{k} b_k X^{n+1-k} \\ &= \frac{b_{n+1}}{(n+1)!} + \frac{1}{(n+1)!}\sum_{k=0}^{n} \binom{n+1}{k} b_k X^{n+1-k} \\ &= \frac{1}{(n+1)!}\sum_{k=0}^{n+1} \binom{n+1}{k} b_k X^{n+1-k} \end{aligned}$$

Proposition. Pour tout $n \geq 1$,

$$b_n = -\frac{1}{n+1}\sum_{k=0}^{n-1} \binom{n+1}{k} b_k$$

Démonstration. Soit $n \geq 1$. Intégrons B_n entre 0 et 1. On a

$$\begin{aligned} 0 &= \int_0^1 B_n(x)\,dx \\ &= B_{n+1}(1) - B_{n+1}(0) \\ &= \frac{1}{(n+1)!}\sum_{k=0}^{n+1} \binom{n+1}{k} b_k - \frac{b_{n+1}}{(n+1)!} \\ &= \frac{1}{(n+1)!}\sum_{k=0}^{n} \binom{n+1}{k} b_k \end{aligned}$$

En isolant b_n on obtient le résultat.

1.2 2. Code Python

La fonction `power_dn` prend en paramètres un nombre x (flottant, complexe, etc.) et un entier n. Elle renvoie la $n$ième puissance descendante de x, qui est

$$x^{\underline{n}} = x(x-1)\dots(x-n+1)$$

[2]:
```
def power_dn(x, n):
    p = 1
    for k in range(n):
        p = p * (x - k)
    return p
```

Pour tout $n \in \mathbb{N}$, $n! = n^{\underline{n}}$.

[3]:
```
def fact(n):
    return power_dn(n, n)
```

[4]:
```
print([fact(n) for n in range(11)])
```

```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
```

Pour tous $n, k \in \mathbb{N}$,

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!}$$

[5]:
```
def binom(n, k):
    return power_dn(n, k) // fact(k)
```

[6]:
```
print([binom(6, k) for k in range(7)])
```

```
[1, 6, 15, 20, 15, 6, 1]
```

La fonction `nombres_bernoulli` prend en paramètre un entier N. Elle renvoie la liste $[b_0, \dots, b_N]$.

[7]:
```
def nombres_bernoulli(N):
    bs = [Fraction(1)]
    for n in range(1, N + 1):
        b = 0
        for k in range(n):
            b = b - binom(n + 1, k) * bs[k]
        bs.append(b / (n + 1))
    return bs
```

[8]:
```
n = 14
bs = nombres_bernoulli(n)
for k in range(n + 1):
    print(k, bs[k])
```

```
0 1
1 -1/2
2 1/6
3 0
4 -1/30
5 0
6 1/42
7 0
8 -1/30
9 0
10 5/66
11 0
12 -691/2730
```

```
13 0
14 7/6
```

La fonction `nombres_bernoulli_bis` prend en paramètre un entier N. Elle renvoie la liste $[B_0(0), \dots, B_N(0)]$.

[9]:
```
def nombres_bernoulli_bis(N):
    bs = nombres_bernoulli(N)
    return [bs[k] / fact(k) for k in range(N + 1)]
```

[10]:
```
n = 14
bs = nombres_bernoulli_bis(n)
for k in range(n + 1):
    print(k, bs[k])
```

```
0 1
1 -1/2
2 1/12
3 0
4 -1/720
5 0
6 1/30240
7 0
8 -1/1209600
9 0
10 1/47900160
11 0
12 -691/1307674368000
13 0
14 1/74724249600
```

La fonction `print_poly` affiche les coefficients d'un polynôme dans l'ordre croissant des degrés.

[11]:
```
def print_poly(P):
    n = len(P)
    for k in range(n):
        print(P[k], end='')
        if k < n - 1: print(',', end=' ')
    print()
```

La fonction `poly_bernoulli` prend en paramètre un entier n. Elle renvoie la liste des coefficients du polynôme $ B_n$ dans l'ordre croissant des degrés.

[12]:
```
def poly_bernoulli(n):
    bs = nombres_bernoulli(n)
    return [binom(n, n - k) * bs[n - k] / fact(n) for k in range(n + 1)]
```

[13]:
```
for n in range(9):
    print(n, end=' ')
```

```
    print_poly(poly_bernoulli(n))
```

```
0 1
1 -1/2, 1
2 1/12, -1/2, 1/2
3 0, 1/12, -1/4, 1/6
4 -1/720, 0, 1/24, -1/12, 1/24
5 0, -1/720, 0, 1/72, -1/48, 1/120
6 1/30240, 0, -1/1440, 0, 1/288, -1/240, 1/720
7 0, 1/30240, 0, -1/4320, 0, 1/1440, -1/1440, 1/5040
8 -1/1209600, 0, 1/60480, 0, -1/17280, 0, 1/8640, -1/10080, 1/40320
```

La fonction `poly_bernoulli_bis` prend en paramètre un entier n. Elle renvoie la liste des coefficients du polynôme de Bernoulli-Wikipedia $n!\,B_n$ dans l'ordre croissant des degrés.

[14]:
```
def poly_bernoulli_bis(n):
    bs = nombres_bernoulli(n)
    return [binom(n, n - k) * bs[n - k] for k in range(n + 1)]
```

[15]:
```
for n in range(11):
    print(n, end=' ')
    print_poly(poly_bernoulli_bis(n))
```

```
0 1
1 -1/2, 1
2 1/6, -1, 1
3 0, 1/2, -3/2, 1
4 -1/30, 0, 1, -2, 1
5 0, -1/6, 0, 5/3, -5/2, 1
6 1/42, 0, -1/2, 0, 5/2, -3, 1
7 0, 1/6, 0, -7/6, 0, 7/2, -7/2, 1
8 -1/30, 0, 2/3, 0, -7/3, 0, 14/3, -4, 1
9 0, -3/10, 0, 2, 0, -21/5, 0, 6, -9/2, 1
10 5/66, 0, -3/2, 0, 5, 0, -7, 0, 15/2, -5, 1
```

1.3 3. Tracé des polynômes de Bernoulli

1.3.1 3.1 Évaluer un polynôme en un point

La fonction `eval_poly` prend en paramètres

- La liste des coefficients d'un polynôme P
- Un nombre (entier, flottant, complexe, ...) x

Elle renvoie $P(x)$.

[16]:
```
def eval_poly(P, x):
    y = 0
```

```
    n = len(P)
    for k in range(n):
        y = y + P[k] * x ** k
    return y
```

Calculons $P(3)$ où $P = 2X^2 - X + 1$.

[17]:
```
P = [1, -1, 2]
eval_poly(P, 3)
```

[17]: 16

1.3.2 3.2 Le tracé des $n!\,B_n$

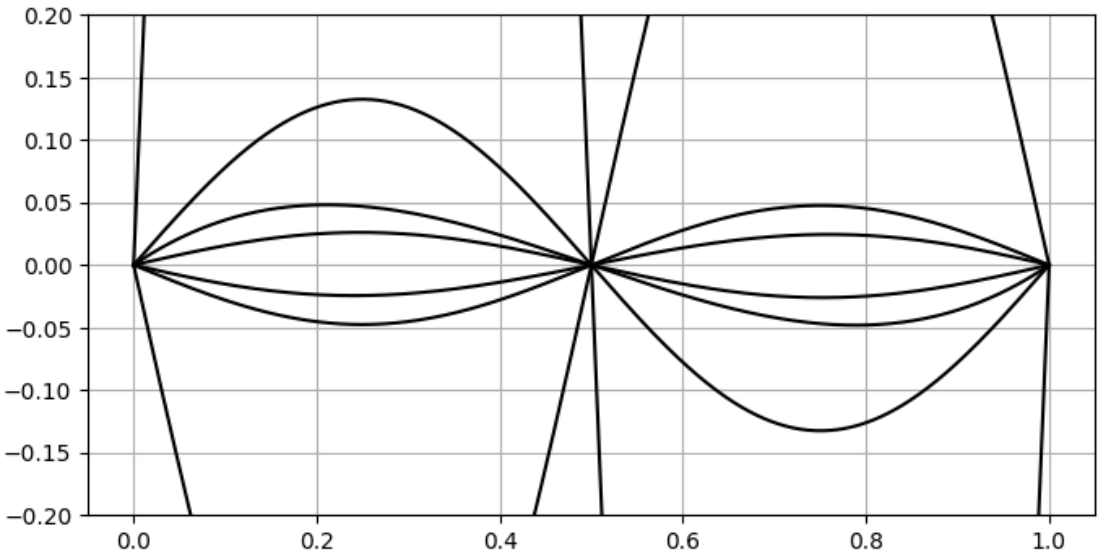
[18]:
```
def subdi(a, b, n):
    d = (b - a) / n
    return [a + k * d for k in range(n + 1)]
```

[19]:
```
def plot_bernoulli(a, b, rng):
    xs = subdi(a, b, 500)
    for n in rng:
        P = poly_bernoulli_bis(n)
        ys = [eval_poly(P, x) for x in xs]
        plt.plot(xs, ys, 'k')
    plt.grid()
```

[20]:
```
plt.rcParams['figure.figsize'] = (8, 4)
```

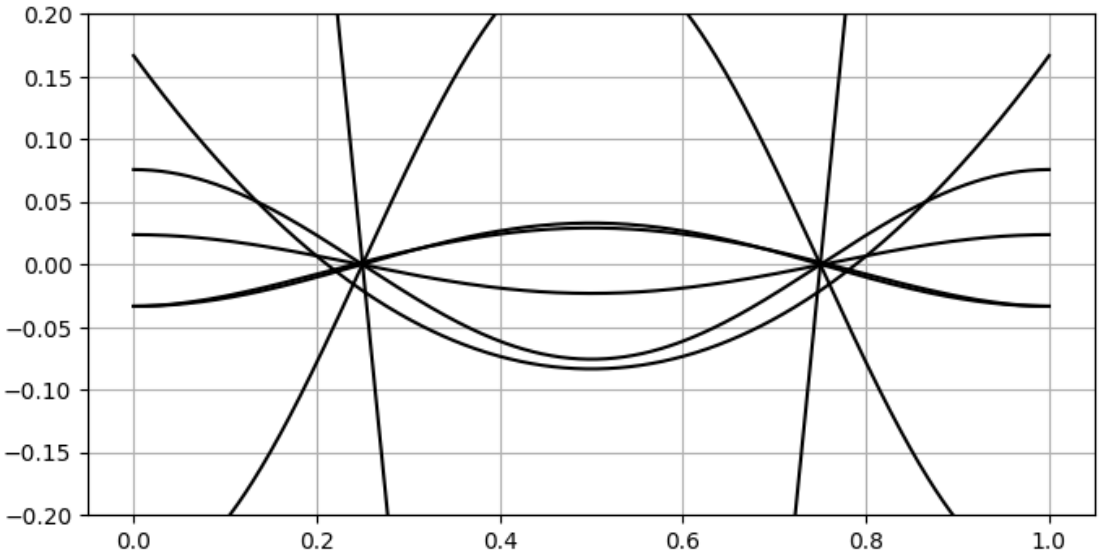
Voici le graphe de $n!\,B_n$ pour $n \in [|3, 15|]$ impair. On peut montrer que pour tout $n \geq 3$ impair, B_n s'annule uniquement en 0, $\frac{1}{2}$ et 1 sur l'intervalle $[0, 1]$.

[21]:
```
plot_bernoulli(0, 1, range(3, 16, 2))
plt.ylim(-0.2, 0.2)
print()
```



Voici le graphe de $n!\,B_n$ pour $n \in [|2, 14|]$ pair. On peut montrer que pour tout n pair supérieur ou égal à 2, B_n a une unique racine α_n sur $]0, \frac{1}{2}[$ et une unique racine β_n sur $]\frac{1}{2}, 1[$. Pour des raisons de symétrie, $\beta_n = 1 - \alpha_n$.

[22]:
```
plot_bernoulli(0, 1, range(2, 15, 2))
plt.ylim(-0.2, 0.2)
print()
```



L'étude de toutes les racines (réelles ET complexes) de B_n conduit à des résultats très intéressants. Voir par exemple à l'adresse

https://arxiv.org/pdf/math/0703452.pdf

Le contenu de l'article pointé par l'adresse n'est pas simple mais les résultats établis sont compréhensibles.

1.4 4. La fonction ζ de Riemann

Soit

$$\mathcal{P} = \{s \in \mathbb{C}, \mathrm{Re}(s) > 1\}$$

Pour tout $s \in \mathcal{P}$, la série de terme général $1/n^s$ est convergente. La fonction $\zeta : \mathcal{P} \longrightarrow \mathbb{C}$ est définie, pour tout $s \in \mathcal{P}$, par

$$\zeta(s) = \sum_{n=1}^{+\infty} \frac{1}{n^s}$$

Pour le lecteur qui ne connaîtrait pas les séries, ceci signifie que pour tout $s \in \mathcal{P}$, la suite $(S_N)_{N\in\mathbb{N}^*}$ définie par

$$S_N = \sum_{n=1}^{N} \frac{1}{n^s}$$

est convergente. La limite de cette suite est ce que l'on note $\sum_{n=1}^{+\infty} \frac{1}{n^s}$.

Proposition. Pour tout $m \in \mathbb{N}^*$,

$$\zeta(2m) = \frac{1}{2}(-1)^{m-1} 2^{2m} \pi^{2m} B_{2m}(0)$$

Démonstration. C'était l'objectif du devoir maison. Rappelons que B_{2m} est ici le polynôme de Bernoulli-DM et pas le polynôme de Bernoulli-Wikipedia. On trouve dans la littérature l'expression de $\zeta(2m)$ en fonction des nombres de Bernoulli :

$$\zeta(2m) = (-1)^{m-1} \frac{2^{2m-1} b_{2m}}{(2m)!} \pi^{2m}$$

La fonction `zeta` prend en paramètre un entier n pair non nul. Elle renvoie

$$\frac{\zeta(n)}{\pi^n}$$

[23]:
```
def zeta(n):
    m = n // 2
```

```
    bs = nombres_bernoulli_bis(n)
    return -(-4) ** m * bs[n] / 2
```

La fonction `str_zeta` prend en paramètre un entier n pair non nul. Elle renvoie une représentation de $\zeta(n)$ sous forme de chaîne de caractères.

[24]:
```
def str_zeta(n):
    return str(zeta(n)) + ' \u03c0' +'^' + str(n)
```

[25]:
```
for n in range(2, 15, 2):
    print('%3d' %n, end=' ')
    print(str_zeta(n))
```

```
 2 1/6  ^2
 4 1/90  ^4
 6 1/945  ^6
 8 1/9450  ^8
10 1/93555  ^10
12 691/638512875  ^12
14 2/18243225  ^14
```

Le lecteur dubitatif constatera sur Wikipedia que ce sont bien les bonnes valeurs. En revanche, il ne trouvera pas sur l'encyclopédie qui contient tout (?) la valeur de $\zeta(234)$. Bien évidemment, cela ne nous pose aucun problème.

[26]:
```
print(str_zeta(234))
```

```
29591829216677027514862380535323793815191086761732488111122588831853966166417199
95581122510566140405469089471655391859660207388424004558662384717306193365438070
74023116830229749442022284200492429746438432891402512004704270335794463181522395
81330085797938826850274842037370 4/637150491263826469189000000786227894185828520 0
85650552454030653722104815209557695490398269893963565521172941530502588319536883
54550921562825024345715043411452781706438589882659314324413344291444342902911150
34672913037116963923686635469445936943538261525299937516456349494601403493449654
84152393326624405789173391642558761800058168591432837724602705212206821272502565
98927080631256103515625  ^234
```

1.5 5. Le comportement asymptotique de B_n

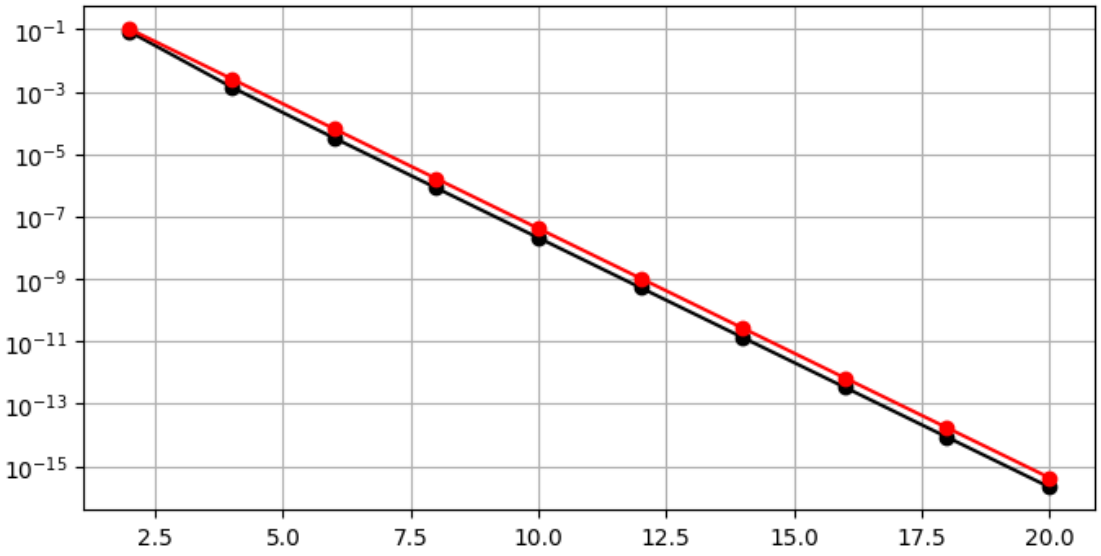
La dernière question du devoir demande de montrer que pour tout $m \geq 1$,

$$|B_{2m}(0)| \leq \frac{1}{4^{m-1}\pi^{2m}}$$

- En noir, les valeurs de $B_{2m}(0)$
- En rouge, le majorant

[27]:
```
def plot_bn(N):
    xs = range(2, N + 2, 2)
    bs = nombres_bernoulli_bis(N)
    ys = [abs(bs[k]) for k in xs]
    zs = [1 / (math.pi ** (k) * 4 ** (k // 2 - 1)) for k in xs]
    plt.semilogy(xs, ys, '-ok')
    plt.semilogy(xs, zs, '-or')
    plt.grid()
```

[28]:
```
plot_bn(20)
```



Clairement, le majorant est bien mieux qu'un « simple » majorant. On peut montrer que

$$B_{2m}(0) \sim \frac{(-1)^{n-1}}{4^{m-1}\pi^{2m}\sqrt{2}}$$

Le majorant trouvé dans le devoir est donc « quasi-asymptotiquement-optimal ». La différence entre notre majorant et l'équivalent de B_n est, sur le graphique en coordonnées logarithmiques,

$$\frac{1}{2}\log_{10} 2$$

Qu'est-ce que cela signifie ? Chaque unité sur l'axe des ordonnées correspond à une multiplication par 10. La différence entre l'ordonnée du point rouge et l'ordonnée du point noir correspondant est environ 0.15.

[29]:
```
math.log(2, 10) / 2
```

[29]:
```
0.15051499783199057
```