TP 18- Proposition de solutions

Solution 1 1. $(11100,1011)_2 = 2^4+2^3+2^2+2^{-1}+2^{-3}+2^{-4} = 28.6875$

2. Écriture binaire de la partie entière utilisant la division successive par 2 :

q_i	28	14	7	3	1	0
r_i	1	0	0	1	1	1
b_i	b_0	b_1	b_2	b_3	b_4	b_5

donc $57 = (111001)_2$
Écriture binaire de la parti fractionnaire avec l'algorithme donné :

f_i	0.321	0.642	0.284	0.568	0.136	0.272	0.544	0.088
$2f_i$	0.642	1.284	0.568	1.136	0.272	0.544	1.088	0.196
b_i	$b_{-1}=0$	$b_{-2}=1$	$b_{-3}=0$	$b_{-4}=1$	$b_{-5}=0$	$b_{-6}=0$	$b_{-7}=1$	$b_{-8}=0$

donc $0.321 = (0,01010010\cdots)_2$
Ainsi, $57,321 = (111001,01010010\cdots)_2$

3. Passage le l'écriture décimale à l'écriture binaire :

```
def binaire_2(x,n):
    e=''.join([str(u) for u in base(int(x))])
    f=x-int(x)
    c=''
    while f>0 and len(c)<n:
        if 2*f>=1:
            c,f=c+'1',2*f-1
        else:
            c,f=c+'0',2*f
    return e+','+c
```

On pouvait astucieusement mutiplier par 2^{n+1} et travailler uniquement sur la nouvelle partie entière puis replacer la virgule au bon endroit :

```
def binaire(x,n):
    L=base(int(x*2**(n+1)))
    c=''.join([str(u) for u in L])
    return c[:-(n+1)]+','+c[-(n+1):-1]
```

4. Passage le l'écriture binaire à l'écriture décimale :

```
def decimal(c):
    v=0
    while v<len(c) and c[v]!=',':
        v=v+1
    if v==len(c): # pas de virgule
        L=[int(u) for u in c]
        return base10(L)
    else:
        c=c[:v]+c[(v+1):]
        L=[int(u) for u in c]
        return base10(L)*2**(v-len(c))
```

Solution 2 La première approche est la moins bonne car en commençant par les grands termes, les petits sont cumulés sur la base de l'imprécision du premier terme.
En revanche, dans la seconde somme, les petits termes sont additionnés ensemble ce qui permet de prendre en compte tous les termes.

```
s,r,N=0,0,10**9
for k in range(1,N):
    s=s+1/k**2
    r=r+1/(N-k)**2
t=np.pi**2/6
print(abs(r-t),abs(s-t))
```

Ce qui donne

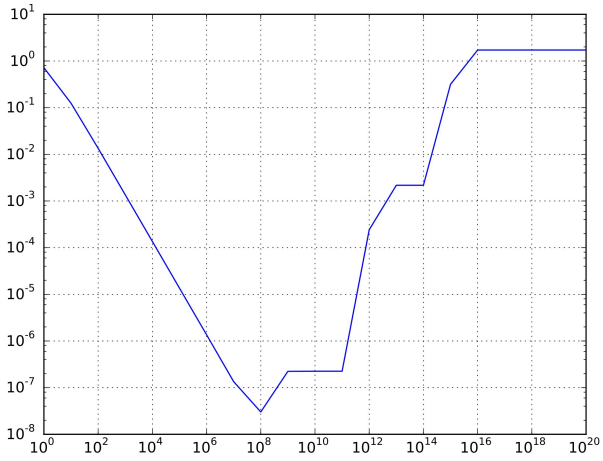
```
1.000000082740371e-09 9.013651380840315e-09
```

Attention ! Pour calculer la somme de beaucoup de termes, mieux vaut commencer par ceux de petites tailles.

Solution 3 1. $\left(1+\frac{1}{n}\right)^n = \exp\left(n\ln\left(1+\frac{1}{n}\right)\right) = \exp\left(n\left(\frac{1}{n}+o\left(\frac{1}{n}\right)\right)\right) = \exp(1+o(1)) \to e$

2. Le script :

```
n=[10**i for i in range(0,21)]
S=[abs((1+1/u)**u-np.exp(1)) for u in n]
plt.plot(n,S)
plt.xscale('log')
plt.yscale('log')
plt.grid()
plt.show()
```



3. Les trois phases sont :
- si $n \leq 10^8$, les calculs sont proches des valeurs exactes et l'approximation augmente avec n
- si $10^8 < n \leq 10^{16}$, l'expression $1+\frac{1}{n}$ est de moins en moins précise. En particulier, le calcul de la puissance par le méthode de l'exponentiation rapide repose sur le calcul de carrés :

$$(1+a)^2 = 1+2a+a^2$$

Le terme en a^2 est complètement absorbé par 1 dès que $a^2 \approx 10^{-16}$ c'est-à-dire lorsque $a \approx 10^{-8}$.

Le terme en $2a$ est progressivement absorbé lorsque a varie de 10^{-8} à 10^{-16}.

- si $n > 10^{16}$, $1+\frac{1}{n}$ est représenté par 1 car $\frac{1}{n}$ est absorbé par 1 et donc la suite est constante à 1 et l'erreur est de l'ordre de l'unité.

Solution 4

1. L'erreur relative est d'un facteur de $2^{-9} \approx 0.002$
2. L'exposant peut prendre des valeurs sur

$$[\![-2^6, 2^6-1]\!] = [\![-64, 63]\!]$$

3. Le plus grand réel est :

```
>>> (2**9-1)*2**(63-8)
18410715276690587648
```

4. Les fonctions de passage de la base 2 à la base 10 :

```
def base2(n):
    L=[n%2] # gestion du cas n=0
    n=n//2
    while n>0:
        L.append(n%2)
        n=n//2
    return L[::-1]

def base10(L):
    n=0
    m=1
    for i in range(len(L)):
        n,m=n+m*L[-i-1],m*2
    return n
```

5. Passer d'un nombre à sa représentation :

```
def bin16(x):
    if abs(x)>(2**9-1)*2**(63-8):
        if x>0: return 'inf'
        else: return '-inf'
    c=0
    while c>-64 and x<2**8:
        c=c-1
        x=x*2
    if c==64:
        return 16*'0'
    else:
        L=base2(int(abs(x)))
        m=''.join([str(u) for u in L[1:9]])
        E=base2(64+len(L)-1+c)
        E=''.join([str(u) for u in E])
        E=(7-len(E))*'0'+E
    if x>0:
        return '0'+m+E
    else:
        return '1'+m+E
```

6. Passer de la représentation au nombre :

```
def reel16(c):
    '''nb represente en precision 16 bits'''
    L=[int(u) for u in c]
    s=1-2*L[0] # (-1)**L[0]
    m=base10([1]+L[1:9])*2**(-8)
    e=base10(L[-7:])-64
    return s*m*2**e
```

Ce qui donne :

```
>>> bin16(12.3)
'0100010011000011'
>>> reel16('0100010011000011')
12.28125
```