Ejercicio 21 y 22

February 4, 2018

Ejercicio 21. Define funciones H(n, nbits) y H2(n, nbits) que devuelvan la suma k=1 k 1 calculada con precisión igual a nbits. Las dos funciones serán esencialmente iguales, pero H realizará todos los cálculos con números decimales de la precisión fijada mientras que H2 debe efectuar los cálculos con racionales, y únicamente al devolver el resultado debe convertir a decimal de nbits de precisión. £Qué enseña este ejemplo?

```
In [13]: def H(n,nbits):
             res = 0
             for i in xsrange (1,n+1):
                 res = res.n(prec=nbits) + (1/i).n(prec=nbits)
             return res.n(prec=nbits)

         def H2(n,nbits):
             res = 0
             for i in xsrange(1,n+1):
                 res = QQ(res) + QQ(1/i)
             return res.n(prec=nbits)

In [14]: print H(100,500)
         print H2(100,500)
         print H(100,500) - H2(100,500)
```

```
5.18737751763962026080511767565825315790897212670845165317653395658721955753255049660568776892
5.18737751763962026080511767565825315790897212670845165317653395658721955753255049660568776892
-4.88789818159936749128316702914178831966304438491572266222978225485006660243955273232297157617
```

Vemos que hay una diferencia mínima entre ambos resultados

Ejercicio 22. 1. Define una función, mi gamma1(n, nbits), que calcule el término n-ésimo de la sucesión que define gamma con nbits de precisión. 2. Define una función, mi gamma2(n, nbits), que calcule

con nbits de precisión. La notación dxe indica la parte entera por exceso de x. El lmite, cuando n tiende a infinito, de n se sabe que es . 3. Define una función, mi gamma3(n, nbits), que calcule gamma sumando n términos de la serie que también converge a . 4. Compara la cantidad de cifras correctas obtenidas y los tiempos de cálculo para los tres métodos usando n = 10 6 y fijando la precisión óptima mediante experimentación. £Qué conclusiones obtienes y cuál puede ser el motivo?

```
In [40]: def Hn(n,nbits):
             res = 0
             for i in xsrange(1,n+1):
                 res = res.n(prec=nbits) + (1/i).n(prec=nbits)
             return res.n(prec=nbits)
         def mi_gamma1(n,nbits):
             return (Hn(n,nbits) - log(n)).n(prec=nbits)
         def mi_gamma2(n,nbits):
             res = 0
             for i in xsrange (1,n+1):
                 res = res + (1/n)*(ceil(n/i)-(n/i))
             return res.n(prec=nbits)
         def mi_gamma3(n,nbits):
             res = 0
             for i in xsrange(1,n+1):
                 res = res + (1/i - log(1 + 1/i))
             return res.n(prec=nbits)

In [44]: print mi_gamma1(10000,100)
         print mi_gamma2(10000,100)
         print mi_gamma3(10000,100)
         print euler_gamma.n(prec=100)

0.57726566406819952810651208572
0.57669396395561773582152209515
0.57716566906786621977117891948
0.57721566490153286060651209008


In [47]: %%time
         mi_gamma1(10000,100)

CPU times: user 1.32 s, sys: 4 ms, total: 1.32 s
Wall time: 1.31 s


Out[47]: 0.57726566406819952810651208572

In [45]: %%time
         mi_gamma2(10000,100)

CPU times: user 140 ms, sys: 0 ns, total: 140 ms
Wall time: 127 ms


Out[45]: 0.57669396395561773582152209515

In [46]: %%time
         mi_gamma3(10000,100)
```

```
CPU times: user 7.23 s, sys: 68 ms, total: 7.3 s
Wall time: 7.19 s
```

```
Out[46]: 0.57716566906786621977117891948
```

Como podemos ver, el primer método es el que más cifras acierta mientras que el que menos es el segundo. Sin embargo, vemos que el segundo tarda muy poco respecto a los otros dos (sobre todo al tercero).